2nd Edition

A Guide to Programming in Applesoft®

Lawrenceville Press

BRUCE PRESLEY

Lawrenceville
Press

A Guide to Programming in Applesoft®

TRADE EDITION

BRUCE PRESLEY

DISTRIBUTED BY
VNR VAN NOSTRAND REINHOLD COMPANY
NEW YORK CINCINNATI TORONTO LONDON MELBOURNE

First edition published in 1982
Second edition published in 1984
Copyright © 1984
by

Lawrenceville Press

Trade Edition
ISBN 0-442-27249-9

All rights reserved. No part of this work covered by the copyright may be reproduced or used in any form or by any means—graphics, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without written permission of the publisher, with the exception of programs, which may be entered, stored and executed in a computer system, but not reprinted for publication or resold in any other form. Printed in the United States of America.

VNR

Educational orders may be placed by writing or calling:

Van Nostrand Reinhold Inc.
7625 Empire Drive
Attn: Department T
Florence, Kentucky 41042
Telephone 606-525-6600

Van Nostrand Reinhold Company
135 West 50th Street, New York, NY 10020

Macmillan of Canada
Division of Gage Publishing Limited
164 Commander Boulevard
Agincourt, Ontario M1S 3C7, Canada

Van Nostrand Reinhold Australia Pty. Ltd.
480 Latrobe Street
Melbourne, Victoria 3000, Australia

Van Nostrand Reinhold Company Ltd.
Molly Millars Lane
Wokingham, Berkshire, England RG11 2PY

16 15 14 13 12 11 10 9 8 7 6 5 4 3

This guide is written and published by Lawrenceville Press, Inc. and is in no way connected with Apple Computer Inc. Apple, Apple II, Apple II Plus, Apple //e, and Applesoft are registered trademarks of Apple Computer Inc. with regard to any computer product.

Acknowledgment

The publication of this second edition of A GUIDE TO PROGRAMMING IN APPLESOFT represents the latest achievement of the staff at Lawrenceville Press in developing curricular materials to assist students in learning to program computers. In addition to all the features that made the first edition so popular with thousands of secondary schools and colleges, this edition also offers new chapters on programming techniques and advanced topics. Because of our classroom experience over the past fifteen years teaching programming at The Lawrenceville School, we are convinced that a structured, concisely written computer text is invaluable in teaching and learning programming.

The production of this manual has involved the cooperation and talents of many individuals. Lester Waters, Robert Lynch, Eli Hurowitz and Donald Mikan have all made invaluable contributions to this publication. Each has authored and taken responsibility for specific areas of the manual. The clarity and accuracy of each chapter reflects their considerable effort and expertise. Marjorie Vining, Josef Helfenbein, and Gregg Schwinn have produced the imaginative graphic designs. James Adams, of The Lawrenceville School, has edited the text to ensure that it is both grammatically and stylistically correct. A very special thanks is due to Archie and Christine Browne and the staff at Browne Book Composition Inc. and the staff at Heffernan Press, Inc.

To all of these people I am deeply grateful. Thanks to their combined efforts we have produced what I believe to be one of the finest computer texts available today.

Bruce Presley

Table of Contents

ACKNOWLEDGMENTS

INTRODUCTION

What is a computer? ix
What is a program? ix
The development of the computer ix
Computer Languages x

CHAPTER ONE—Introduction to Programming

Line Numbering 1.1
PRINT 1.1
Typing Errors 1.2
LIST 1.2
NEW 1.3
What is a variable? 1.3
Numeric Variables 1.3
String Variables 1.5
READ, DATA 1.7
GOTO 1.7
INPUT 1.9
Print Formatting 1.10
PRINT TAB 1.12
Immediate Mode 1.14
REM 1.15
Exercises 1.16

CHAPTER TWO—Decisions and Loops

Order of Operations 2.1
IF... THEN 2.1
Extended Use of the IF... THEN Statement, Multiple Statement Lines 2.3
AND, OR 2.5
END 2.6
FOR... TO... STEP, NEXT 2.7
RESTORE 2.10
Exercises 2.13

CHAPTER THREE—Computer Games

RND . . . 3.1
INT . . . 3.1
Summation . . . 3.3
Rounding Errors . . . 3.4
POS . . . 3.5
HTAB, VTAB, and HOME . . . 3.6
FLASH, INVERSE, and NORMAL . . . 3.6
GET . . . 3.8
Game Programs . . . 3.9
Exercises . . . 3.14

CHAPTER FOUR—Nested Loops and Subscripted Variables

Nested FOR . . . NEXT Loops . . . 4.1
The Need for Subscripted Variables . . . 4.3
Single Subscripted Variables . . . 4.4
DIM . . . 4.9
CLEAR . . . 4.9
Double Subscripted Variables . . . 4.10
Some Final Notes on Subscripted Variables . . . 4.13
Extended Variable Names . . . 4.13
Exercises . . . 4.14

CHAPTER FIVE—Programming Techniques

GOSUB, RETURN . . . 5.1
ON . . . GOTO . . . 5.2
ON . . . GOSUB . . . 5.4
Planning a Program . . . 5.5
Debugging . . . 5.12
Runtime Errors . . . 5.12
Logic Errors . . . 5.13
ONERR GOTO and RESUME . . . 5.14
Hand Tracing . . . 5.17
TRACE and NOTRACE . . . 5.18
STOP and CONT . . . 5.19
Exercises . . . 5.21

CHAPTER SIX—Graphics

Low Resolution Graphics . . . 6.1
GR . . . 6.1
TEXT and HOME . . . 6.1
COLOR . . . 6.2
PLOT . . . 6.2
HLIN . . . 6.3
VLIN . . . 6.3
SCRN . . . 6.6
High Resolution Graphics . . . 6.9
HGR and HGR2 . . . 6.10

HCOLOR 6.10
HPLOT 6.10
Switching Displays 6.13
Exercises 6.17

CHAPTER SEVEN—Mathematical Functions

SQR 7.1
SGN and ABS 7.1
Trigonometric Functions: SIN, COS, TAN 7.3
ATN 7.4
Logarithms and the Exponential Functions: LOG, EXP 7.5
DEF 7.6
Shifting and Scaling a Graph 7.8
Exercises 7.11

CHAPTER EIGHT—String Functions and Data Types

Binary Code 8.1
Computer Memory and Processing 8.1
Data Types 8.4
ASCII Code—ASCII Character Conversion 8.5
String Manipulation Functions 8.6
Exercises 8.9

CHAPTER NINE—Files

Format of File Commands 9.1
OPEN (sequential files) 9.2
CLOSE 9.3
WRITE (sequential files) 9.3
READ (sequential files) 9.5
Updating Sequential Files, APPEND 9.6
Demonstration Programs 9.10
Random-Access Files 9.13
OPEN (random-access files) 9.14
WRITE (random-access files) 9.14
READ (random-access files) 9.16
Demonstration Programs 9.18
Multiple File Use 9.22
Exercises 9.24

APPENDIX A—Editing

Cursor Movement A.1
CONTROL-X A.1
ESC A, B, C or D A.1
CONTROL-S A.1
SPEED A.1
Starting Programs at Arbitrary Locations A.2

APPENDIX B—Cassette Operation

Saving a Program, SAVE B.1
Recalling a Program, LOAD B.1
Storing Data, STORE B.1
Recalling Data, RECALL B.1
Storing Strings on Tape B.2

APPENDIX C—Diskette Operation

Proper Care of Equipment C.1
Making a New Diskette Operational, INIT C.1
SAVE C.1
LOAD C.2
CATALOG C.2
RENAME C.2
DELETE C.3
LOCK and UNLOCK C.3
MAXFILES C.4
Multiple Drive Systems C.4

REVIEW ANSWERS R.1

ANSWERS TO ODD NUMBERED EXERCISES E.1

INDEX

Introduction

When discussing computers, it is convenient to divide the subject into two broad categories—hardware and software. Hardware refers to the computer itself and its peripheral devices—disk drives, printers, etc., whereas software refers to the programs or instructions which are entered into the computer to make it perform specific tasks. It is the purpose of this book to teach you how to program your computer. Therefore, it is a guide for producing software.

What is a computer?

A computer is a machine that accepts information, processes it according to specific instructions, and provides the results as new information. The computer differs from a simple calculator in several ways. It can store and move large quantities of data at very high speed and, even though it cannot think, it can make simple decisions and comparisons. For instance, the computer is capable of determining which of two numbers is larger or which of two names comes first alphabetically. Another major difference is that the computer can perform many varied tasks by changing its instructions. As you progress through this book, you will learn how to instruct the computer to perform increasingly sophisticated tasks by writing programs.

What is a program?

A program is a list of instructions that direct the computer in performing its operations. The computer never does any of its own thinking but merely repeats operations that it has been instructed to do. It is the program that instructs the computer as to which operations it will perform and in what sequence it will perform them.

The Development of the Computer

The modern computer evolved from simple calculating machines. Although for centuries simple arrangements of beads, wheels, and rods had been employed to perform addition and subtraction, the first large scale effort to design a machine capable of performing higher mathematical functions was the "Difference Engine" of Charles Babbage, first produced in 1820, which used a complicated system of gears and wheels. The British government later funded a project to make a large scale version of Babbage's machine, but the project was never successfully completed.

The first major breakthrough on the way to producing a modern computer occurred in the early nineteen forties when engineers started to use vacuum tubes to replace mechanical elements in calculating machines. This innovation resulted in the first fully electronic machine called the ENIAC (Electronic Numerical Integrator And Calculator), which was built at the University of Pennsylvania in 1946. Although considerably faster than its mechanical predecessors, the ENIAC had the major drawback of having its programs wired into the machine. Huge calculations might take only minutes to solve, but programming the machine could take many days.

John von Neumann proposed the idea of storing a program in a computer's memory rather than wiring it. This would eliminate the problem of rewiring the machine each time a new program was needed. The first computer to use the von Neumann concept was the EDSAC (Electronic Delay Storage Automatic Calculator) built at Cambridge University in 1949. Other computers quickly followed with increased speed and efficiency.

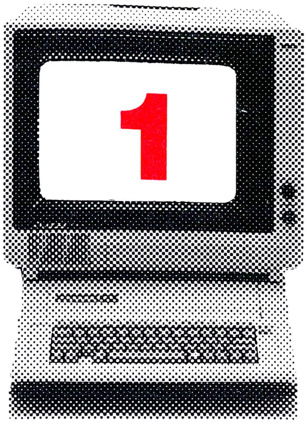
The next significant development occurred in the early nineteen sixties with the introduction of transistors into computers to replace the large and inefficient vacuum tubes. This innovation substantially decreased the size of a computer while increasing its speed and efficiency. But it is only within the last ten years with the development of integrated circuits to replace most of the computer's components that the computer has decreased both in size and in price to the point where it is now available to almost anyone desiring one. The computer you will program is considerably more powerful and efficient than the early computers built only twenty-five years ago, yet it costs only a tiny fraction of the early machines.

Computer Languages

The first modern computers used a group of commands called machine language. A command in machine language is just a group of specially coded numbers that the computer understands. This language is still used today within the computer, but few people ever program in it. Just as computers advanced, so did programming languages. Today there are hundreds of higher level programming languages with names like FORTRAN, BASIC, PASCAL, and PL/I. The computer translates the higher level programming language into machine language and then executes the program. The purpose of this manual is to present a comprehensive guide designed to teach the beginner the programming language called BASIC (Beginners All-purpose Symbolic Instruction Code), developed in the early nineteen sixties by John Kemeny and Tim Kurtz at Dartmouth College.

BASIC, as the name suggests, is well suited for the beginner. The commands are easy to remember because they describe the operations they represent. Like any programming language, BASIC has its specific commands and rules of syntax which determine the form of the commands submitted to the computer. In addition, the computer will only understand commands defined in the higher level language it is using. For instance, in the language BASIC the command PRINT will direct the computer to send information to the screen, but a similar command WRITE will not work at all. This similar command WRITE seems to have the same meaning as PRINT, but the computer will not understand it because WRITE is not defined in the language called BASIC.

This manual will teach you the specific commands and rules necessary to program your microcomputer in BASIC.



INTRODUCTION TO PROGRAMMING

PRINT

LIST

NEW

READ

DATA

GOTO

INPUT

PRINT TAB

REM

Communicating with the computer requires a special language. The language described in this manual is APPLESOFT BASIC, which consists of simple English words understood by the computer. In this chapter the user will be introduced to the most fundamental statements required to write a program on the Apple computer.

Line Numbering

To establish the sequence of instructions for any given program, the programmer must assign numbers to each line within a program. This operation is essential because the computer reads instructions beginning with the lowest line number, then progresses to the next higher number and ends with the highest numbered line.

It is permissible to type the program lines out of order because the computer puts them back into their proper order. For example, line 20 may have been typed before line 10, but when the program is run, the computer automatically reads line 10 first. Such a system has the advantage of allowing any forgotten instructions, say between lines 10 and 20, to be entered later as lines 15, or 12, or 17, etc. For this reason it is best to number the lines of the program in units of ten (10, 20, 30, etc.) since this leaves nine possible line numbers that can later be inserted. It is also important to realize that the capital letter O cannot be used in place of zero.

An error in any instruction may be corrected at any time by depressing the RETURN key and retyping the entire line using the same line number. If two lines are given the same line number, the computer uses only the last one typed and erases the first. Typing only the line number followed by the RETURN key eliminates the line entirely. The highest line number allowed on the computer is 63999.

PRINT

There are two basic types of information which the computer can use: numbers and words. What distinguishes one from the other is that a number can be used in a mathematical calculation whereas a word cannot. The PRINT statement is used to print both numbers and words.

Program 1.1

This program uses the PRINT statement at each line to print the results shown below:

```
10  PRINT 5 * 3
20  PRINT 12 + 8
30  PRINT 7 / 2
40  PRINT "SAND"
50  PRINT "5*3"

]RUN
15
20
3.5
SAND
5*3

]
```

Line 10 instructs the computer to print the product of 5 and 3. Note than an asterisk (∗) is used to indicate multiplication. Line 20 adds 12 and 8 whereas line 30 divides 7 by 2, with the slash (/) indicating division. At line 40 the word SAND is printed. When a word or any set of characters is printed, it must be enclosed within quotation marks. Line 50 prints "5∗3" rather than 15 because quotation marks are used. Information enclosed in quotation marks is called a string; therefore, lines 40 and 50 contain strings.

The command RUN is typed after a program has been entered to execute the program, but it is not part of the program.

Typing Errors

An error in a program can be corrected in one of three ways. First, the program line can be retyped using the same line number. For example,

```
]40 PRINT "ROCKS"

]RUN
15
20
3.5
ROCKS
5*3

]
```

changes only that part of the output of Program 1.1 which is produced by line 40. The second method of correcting an error uses the keys marked → and ←and may be used if the error is detected while the line containing the error is being typed. Pressing the ← key a sufficient number of times will move the cursor on the screen back to the location of the error, where typing the correct symbol will replace the error. Pressing the → key will bring the cursor to the end of the incomplete line so that typing can resume.

LIST

The LIST command is used to print the program currently in the computer's memory. Typing LIST followed by striking the RETURN key will print the entire program; typing LIST followed by a line number and a RETURN will cause only that line to be printed. To list portions of a program, type LIST followed by the first and last line numbers of the portion desired, separating the two numbers with a hyphen (–). For example:

```
]LIST

10  PRINT 5 * 3
20  PRINT 12 + 8
30  PRINT 7 / 2
40  PRINT "ROCKS"
50  PRINT "5*3"
```

```
]LIST 50

50  PRINT "5*3"

]LIST 20-40

20  PRINT 12 + 8
30  PRINT 7 / 2
40  PRINT "ROCKS"


]
```

NEW

Before entering a program into the computer the programmer should use the NEW command to clear the computer's memory. The use of this command insures that program lines from a previous program do not affect the new program:

```
]LIST

10  PRINT 5 * 3
20  PRINT 12 + 8
30  PRINT 7 / 2
40  PRINT "ROCKS"
50  PRINT "5*3"

]NEW

]LIST


]
```

Observe that after the NEW command is used, a subsequent LIST shows that no program is currently in the computer's memory.

What is a Variable?

One of the most useful properties of a computer is its ability to manipulate variables. A variable is a name which represents a value. On the computer, as in mathematics, X, Y, and Z are variables in the expression $Z = 10 + X + Y$ since they can assume many different values. The computer allows words as well as numbers to be represented by variables.

Numeric Variables

Numeric variables are represented by letters or by letters and numbers. A, D, A1, and B1 are all legal names for numeric variables. The computer allows the use of longer names for variables, but for the sake of simplicity only a single letter or a single letter followed by a single digit will be used in the early chapters of this manual.

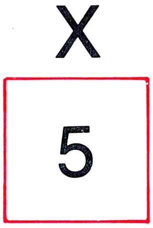
To assign a value to a variable the following format is used:

<variable name> = <value>

For example, to assign X the value 5, the statement

```
20   X = 5
```

could be used. Thus, when line 20 is reached, the variable X will be assigned the value 5. Internally, the computer stores values in memory locations. It is possible to compare these memory locations to boxes. In this case X is the name of the box, and 5 is the value in the box.

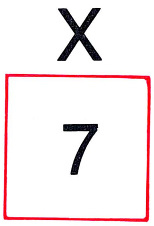


Variables can change in value as the name itself implies; however, it is important to realize that a variable can hold only one value at a time. Suppose a later statement such as

```
50   X = 7
```

is entered. Then at line 50 the value of X will change from 5 to 7.

Thus, the value in the box named X is now 7:



The old value of X is lost and may not be retrieved again.

One numeric variable can be defined in terms of another in the computer, just as in algebra. For example, when the statement Y = 4*X is executed by the computer, Y will be assigned the value 20 if X = 5, or Y will be assigned 28 if X = 7, and so on.

It is important to realize that the equal sign (=) is interpreted as "is assigned the value." Therefore, when assigning a value to a variable, the statement must be in the form: 'variable = value', *not* 'value = variable'. Therefore, the statement

```
50   7 = X
```

is invalid.

Program 1.2

This program assigns values to the variables X and Y, the value of Y being dependent upon the value of X:

```
10 X = 12
20 Y = 3 * X + 5
30  PRINT X,Y
40 X = 15
50  PRINT X,Y
60 Y = 3 * X + 5
70  PRINT X,Y

]RUN
12              41
15              41
15              50

]
```

The value of X is originally 12 at line 10 and then becomes 15 at line 40. Y changes its value at line 60 because X changed at line 40. Note than when X and Y are printed at line 50, X is now 15, but Y is still 41 because the variable Y is not changed until it is reassigned at line 60. A comma is used in the PRINT statements at lines 30, 50, and 70 to allow the values of X and Y to be printed on a single line. On the Apple up to three variables can be printed on a single line of the display by the use of commas to separate each variable name.

String Variables

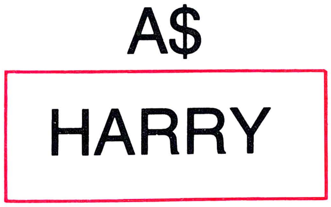
String variables are variables which are assigned values consisting of a sequence of characters. A character is a symbol found on the computer keyboard, which may be a letter, a number or any of the special symbols.

String variable names are used in the same way as numeric variable names, except that a string variable name must end with a dollar sign ($). For example, A$, D$, A1$, and C5$ all represent string variables. To assign characters to a string variable, the programmer must enclose the characters in quotation marks. For example, the statement

10 A$ = "HARRY"

will assign "HARRY" to the string variable A$.

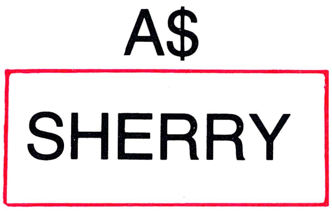
If the box analogy is used again, memory can be thought of as follows:



At some later point in a program it is possible to assign different characters to the same variable. For example, the statement

50 A$ = "SHERRY"

will replace "HARRY" with "SHERRY." Thus, the box named A$ now contains "SHERRY."



Program 1.3

This program assigns two different sets of characters to the string variable B$:

```
10 B$ = "GEORGE"
20  PRINT B$
30 B$ = "JUDY"
40  PRINT B$
50  PRINT "IT IS NICE TO SEE YOU, ";B$

]RUN
GEORGE
JUDY
IT IS NICE TO SEE YOU, JUDY

]
```

Observe that at line 50 a sentence can be formed from two strings, the first of which in this case is not a variable.

Note that when a semicolon rather than a comma is used to separate items in a PRINT statement, the items will be printed next to each other on the same line.

Review

1. Write a program that will print the value of Y when X = 5 and Y = 5X + 7.

2. Write a program that will produce the following output by using string variables for "HARRY" and "SHERRY" but not for the other words.

```
]RUN
HELLO, HARRY
SHERRY IS LOOKING FOR YOU.

]
```

READ, DATA

A variable may be directly assigned its value not only by a statement such as X = 5 but also by a combination of READ and DATA statements.

Program 1.4

This program uses READ and DATA statements to assign values to numeric variables:

```
10  READ X,Y,Z
20  PRINT X + Y + Z
30  DATA  3,2,0.5

]RUN
5.5

]
```

The READ statement at line 10 instructs the computer to assign numbers to the variables X, Y, Z. The computer finds these numbers in the DATA statement at line 30 and assigns them in the order listed (X = 3, Y = 2, and Z = 0.5). If the program contains more than one DATA statement, the computer starts reading data from the first statement and continues sequentially through the remaining DATA statements.

In a DATA statement only numbers that are expressed in decimal or scientific form are acceptable. For example, the numbers .0032, -5.78, 1050, 2.9E5, and 4.76 E-12 are acceptable. The E stands for "times ten to the power" (e.g., 5.3 E3 = 5300 and 8.72 E−3 = .00872). However, data such as the arithmetic expressions 15/3, 3*5, and 7+2 are not permitted in DATA statements since, unlike PRINT statements, DATA statements allow for no calculations.

GOTO

Suppose that a student has more than one set of values for the variables X, Y, Z in Program 1.4. To introduce the extra values, the student can retype line 30 as follows:

```
]30 DATA 3, 2, 0.5, -7, 2.5, 1E2

]RUN
5.5

]
```

Note that only the first three numbers in the DATA statement are processed; the remaining numbers are not used. To overcome this problem, the programmer can retype line 30 each time a new set of data is to be used, but this involves unnecessary labor. A GOTO statement between lines 20 and 30 solves this problem by establishing a loop which allows line 10 to be used again.

```
]LIST

10   READ X,Y,Z
20   PRINT X + Y + Z
25   GOTO 10
30   DATA  3, 2, 0.5, -7, 2.5, 1E2

]RUN
5.5
95.5

?OUT OF DATA ERROR IN 10
]
```

Observe that by typing the command LIST the computer prints the current version of Program 1.4 including the new line. Running Program 1.4 now allows all the data in line 30 to be processed.

Each time the computer reaches line 25 the GOTO statement causes the computer to return to line 10 where the next set of data is assigned to the variables. On the first pass through the loop X =3, Y = 2, Z = .5; on the second pass X = −7, Y = 2.5, Z = 1E2. However, on the third attempted pass, no additional data is available for assignment to the variables at line 10 and, therefore, an error message is printed.

The location of DATA statements within a program is not important, so they can be placed anywhere. Good programming style suggests that you place them at or near the end of a program. When the computer encounters a READ statement, it makes use of DATA statements regardless of their location. DATA statements may contain strings as well as numbers.

Program 1.5

This program reads the names and grades of three students and prints the averages of their grades:

```
5  PRINT "NAME","AVERAGE"
10  READ N$,A,B,C,D
20 X = (A + B + C + D) / 4
30  PRINT N$,X
40  GOTO 10
50  DATA  WATERS,83,95,86,80,KANE,56,97,66,89,MIKAN,61,83,42,90

]RUN
NAME            AVERAGE
WATERS          86
KANE            77
MIKAN           69

?OUT OF DATA ERROR IN 10
]
```

Examine this program carefully because it contains a number of important concepts. Line 5 produces the headings for the columns and is placed at the beginning of the program to insure that the headings will be printed only once. Line 10 assigns a student's name to the string variable N$ and the student's four grades to the numeric variables A, B, C, D. Unless the sequence of string variables and numeric variables in the READ statement is the same as the sequence of strings and numbers in the DATA statement, an error will occur. Line 40 returns the program to line 10 to read more data. What would happen if

```
40 GOTO 5
```

were substituted for the current line 40?

Review

3. Write a program using READ, DATA statements to evaluate Y where Y = 3X + 5 and X = 3, 5, 12, 17, 8.

4. Write a program containing the DATA line from Program 1.5 which prints only the name and first grade of each student.

```
]RUN
NAME                FIRST GRADE
WATERS              83
KANE                56
MIKAN               61

?OUT OF DATA ERROR IN 10
]
```

INPUT

In many instances it is preferable to introduce data from the keyboard rather than place it in a DATA statement. To do this an INPUT statement is used in place of the READ, DATA statements. When an INPUT statement is executed, the computer prints a question mark (?) and then waits for data to be entered.

Program 1.6

Data entered from the keyboard is used to assign a value to the variable X in the following program:

```
10  INPUT X
20  PRINT 5 * X * X + 3
30  GOTO 10
```

```
]RUN
?3
48
?6
183
?2
23
?

BREAK IN 10
]
```

The GOTO statement at line 30 creates a loop which will continue to run until it is halted. To halt the run of a program press the C key while depressing the CTRL key (marked CONTROL on the Apple //e). This operation will also inform the user at which line the computer was when the halt was done.

Program 1.7

This program is a revision of Program 1.5. Here a student is asked for his or her name and four grades. The computer then prints the name and grade average. It is possible to have the INPUT statement print a question or a remark by enclosing the words to be printed in quotation marks followed by a semicolon and the variable names. Also, note that more than one variable can be entered by using a single INPUT statement:

```
10  INPUT "WHAT IS YOUR NAME? ";N$
20  INPUT "WHAT ARE YOUR GRADES? ";A,B,C,D
30 X = (A + B + C + D) / 4
40  PRINT N$,X
50  GOTO 10

]RUN
WHAT IS YOUR NAME? TED
WHAT ARE YOUR GRADES? 87, 54, 76, 95
TED             78
WHAT IS YOUR NAME? JOHN
WHAT ARE YOUR GRADES? 98, 96, 47, 84
JOHN            81.25
WHAT IS YOUR NAME? ALBERT
WHAT ARE YOUR GRADES? 100, 95, 31, 60
ALBERT          71.5
WHAT IS YOUR NAME?

BREAK IN 10
]
```

PRINT Formatting

There are a number of ways in which output can be formatted by the use of punctuation. A few of the more important uses of punctuation are demonstrated by the following program.

Program 1.8

The Apple is capable of printing 40 characters on a single line. This line is divided into three printing zones. The first two zones consist of 16 characters and the last zone holds 8 characters. When commas are used in a PRINT statement the output is printed in successive zones. However, if commas are used and 8 more characters are printed in the second zone, the third zone will be skipped over and output will continue starting at the beginning of the next line. Note that this is different if the Apple is used with an 80 column adapter card.

```
5  PRINT "NAME","GRADES","AVERAGE"
10  READ N$,A,B,C
20 X = (A + B + C) / 3
30  PRINT
40  PRINT N$,A;" ";B" ";C;"          ";X
50  GOTO 10
60  DATA   HAYES,90,87,93,GRAHAM,74,98,83,WATSON,53,76,18
```

```
]RUN
NAME            GRADES          AVERAGE

HAYES           90 87 93         90

GRAHAM          74 98 83         85

WATSON          53 76 18         49

?OUT OF DATA ERROR IN 10
]
```

Note the output produced by line 5. Each word begins at the beginning of one of the zones because commas have been used to separate the words. The PRINT at line 30 is used to place a blank line between each of the printed lines.

When semicolons (;) are used in place of commas (as in line 40), the output of each variable begins in the next space following the last character printed. Therefore, it is necessary to print a space between each variable in order to separate them.

Program 1.9

This program demonstrates the difference between commas and semicolons used in a PRINT statement:

```
10  PRINT "ZONE 1","ZONE 2","ZONE 3"
20 X =  - 14
30  PRINT "**";X;"**"
40 T = 42
50  PRINT "**";T;"**"
60 A$ = "86"
70  PRINT "**";A$;"**"
80  PRINT X;" ";T;" ";A$;99
90 E$ = "ABCDEFGH"
100  PRINT E$,E$,E$
```

```
]RUN
ZONE 1          ZONE 2          ZONE 3
**-14**
**42**
**86**
-14 42 8699
ABCDEFGH        ABCDEFGH
ABCDEFGH

]
```

Note how the values −14 and 42 were printed by lines 30 and 50, respectively. Line 100 demonstrates what happens when 8 characters are printed in zone 2.

PRINT TAB

The PRINT TAB statement provides an easy way to format output and allows the programmer to begin portions of the printout at specified locations. The left edge of the Apple screen is TAB(1), and the right edge of the screen is TAB(40). The printing of information at TAB positions 13, 25, and 32 is accomplished in the following manner:

```
10  PRINT  TAB( 13);"THIS"; TAB( 25);"IS"; TAB( 32);"TAB"

]RUN
            THIS        IS     TAB

]
```

It is important to use semicolons in each instance after the TAB parentheses.

Program 1.10

The substitution of variables for the numbers in the TAB parentheses is permissible provided these variables have previously been assigned values.

```
10  READ X,Y,Z
20  PRINT  TAB( X);"THIS"; TAB( Y);"IS"; TAB( Z);"TAB"
30  DATA  13,25,32

]RUN
            THIS        IS     TAB

]
```

Program 1.11

This program draws a triangle on the display screen:

```
10  PRINT  TAB( 10);"*******"
20  PRINT  TAB( 11);"*"; TAB( 16);"*"
30  PRINT  TAB( 12);"*"; TAB( 16);"*"
40  PRINT  TAB( 13);"*"; TAB( 16);"*"
50  PRINT  TAB( 14);"*"; TAB( 16);"*"
60  PRINT  TAB( 15);"*"; TAB( 16);"*"
70  PRINT  TAB( 16);"*"
```

```
]RUN
         *******
          *    *
           *   *
            *  *
             * *
              **
               *

]
```

The PRINT TAB statement moves the cursor to the right only, never to the left. If a TAB position is specified which is to the left of the current TAB position, the cursor will remain where it is and the printing will proceed from there.

```
10  PRINT  TAB( 10);"##"; TAB( 15);"##"
20  PRINT  TAB( 15);"##"; TAB( 10);"##"
```

```
]RUN
         ##   ##
              ####

]
```

Review

5. Write a program in which you input a value of X and have the computer calculate 5*X and X/5. The printout should appear exactly as shown below:

```
]RUN
WHAT IS X? 12
5*X = 60
X/5 = 2.4

WHAT IS X? 20
5*X = 100
X/5 = 4
```

```
WHAT IS X? 64
5*X = 320
X/5 = 12.8

WHAT IS X?

BREAK IN 10
]
```

6. Write a program which allows you to input your name and the name of a friend and then produces the printout as follows:

```
]RUN
WHAT IS YOUR NAME? BRUCE
WHAT IS YOUR FRIEND'S NAME? JUDY
JUDY IS A FRIEND OF BRUCE

WHAT IS YOUR NAME? DON
WHAT IS YOUR FRIEND'S NAME? SHERRY
SHERRY IS A FRIEND OF DON

WHAT IS YOUR NAME?

BREAK IN 10
]
```

Immediate Mode

The computer may perform simple tasks using the immediate mode rather than a program. An immediate mode instruction is typed without a line number. When the computer receives a command without a line number, it recognizes that the command is not part of a program but is to be executed immediately. The following are examples of immediate mode statements:

```
]PRINT (3*5) + 4
19

]PRINT 7/9
.777777778

]A=5

]B=2

]PRINT A + B + 4
11

]A$="BETTY"

]PRINT A$;" BOO"
BETTY BOO

]
```

Note that a RUN command was not used in any of the preceding examples. Most commands are permissible in the immediate mode with the notable exception of the INPUT statement. In performing most tasks it is best to write complete programs and to reserve the use of immediate mode for simple calculations.

REM

The REM statement is used in the body of a program to allow the programmer to introduce explanatory remarks. Everything to the right of a REM statement is ignored by the computer when the program is run. For example,

```
30 REM CALCULATE AREA OF CIRCLE
```

will be printed only when the program is listed. Remarks placed strategically within a program are useful in explaining the function of various parts of the program.

Program 1.12

This program incorporates the major topics covered in this chapter, which the reader should now be familiar with, and demonstrates how they might be used together in a single program.

```
10   REM  This program demonstrates the material in chapter 1
20   PRINT "HOW FAR OVER IS EACH ITEM"
30   INPUT "TO BE PRINTED? ";T
40   READ A$
50   PRINT  TAB( T);A$
60   GOTO 40
70   DATA  THIS,IS,A,SAMPLE,OF,THE
80   DATA  MATERIAL,IN,CHAPTER,ONE

]RUN
HOW FAR OVER IS EACH ITEM
TO BE PRINTED? 7
      THIS
      IS
      A
      SAMPLE
      OF
      THE
      MATERIAL
      IN
      CHAPTER
      ONE

?OUT OF DATA ERROR IN 40
]
```

Exercises

1. Write a program which prints the following:

```
]RUN
A
 B
  C
ABCD

]
```

2. Write a program which makes the following calculations. Check the results by hand computation:

   (a) 8 * (19 - 4)
   (b) 1 * 2 + 2 * 3 + 3 * 4 + 4 * 5
   (c) 1 + 1 / 10 + 1 / 100
   (d) 10 - 20 + 30 - 40 + 50 - 60

3. What is the exact output for the following program?

```
10 A = 3
20  PRINT "The value of B"
30 B = A + 4 * 4
40  PRINT B
```

4. How many lines of output does the following program produce before an error message appears indicating "?OUT OF DATA ERROR IN 10"?

```
10  READ A,B,C
20  PRINT (A + B + C) / 3
30  GOTO 10
40  DATA  11,32,42,14,25,36,47
50  DATA  58,39,50,61,22,83,94
```

5. Peter Goodpepper has loused up once again. Here is a program of his which he says "mysteriously doesn't work." Give Peter a hand and correct this monstrosity for him so that the output looks like the following:

```
THE SUM IS 20
THE SUM IS 14
?OUT OF DATA ERROR IN 10
```

```
10   READ AB
20   PRINT ThesumisA + B
30   GOTO 12
40   DATA  12,8,3*5,5
```

6. Write the output and check by running the program.

```
10   READ A$
20   PRINT A$
30   GOTO 10
40   DATA  S,I,X!," 6",6
```

7. Write a program in which the price (P) in cents of a loaf of bread and the number (N) of loaves bought are entered from the keyboard. The total spent for bread is to be printed in dollars and cents.

8. Write a program which will enter your weight (W) in pounds and height (H) in inches and which then will print the quotient W/H followed by the words "POUNDS PER INCH."

9. Using an INPUT statement, write a program which produces the following output:

```
]RUN
?2,4
X = 2              Y = 4              X*Y = 8
?-8,70
X = -8             Y = 70             X*Y = -560
?
```

10. Predict the output of the following program. Check the answer by running the program:

```
10 P$ = "Total price"
20 P = .89
30  PRINT P$;" = $0";P
```

11. Predict the output and check by running the program:

```
10  READ A,B,C,D
20  PRINT A,B
30  PRINT C;"   ";D
40  DATA  3E2,510,3E-1,.51
```

12. Use immediate mode to print the following numbers:

(a) .05	(b) .005
(c) 123456789	(d) 1234567890
(e) −.065	(f) 4.62E5
(g) .086E-3	(h) −4,900

13. Self-proclaimed computer whiz Cecil Robert Clancy has typed the following gibberish in the immediate mode and has challenged anyone to guess the output correctly before he or she presses the RETURN key. Put Cecil in his place and correctly predict the answer.

```
]PRINT "AAA";111,222;"AAA","333";"  ";16-3*2
```

14. Have the computer produce the following output:

```
]RUN
1234567890123456789012345678901234567890
*       9     ?     -       !   )      *

]
```

15. What is the exact output for the following program?

```
10 A$ = "abcd"
20 B$ = "xyz"
30 F = 7
40 G =  - 4
50  PRINT A$;B$
60  PRINT A$;F
70  PRINT F;B$
80  PRINT G;B$
```

16. Write a program to evaluate each of the following expressions for A = 10 and B = 7:

$(7A + 10B)/2AB$ $\qquad$ $(1/2)A/(A-B)$

17. In the first week of the season the cross country team ran the following number of miles each day: 2, 3, 4, 3, 5. Write a program to calculate and print the total mileage for the week.

18. A piece of pizza normally contains about 375 calories. A boy jogging one mile uses about 100 calories. Write a program that asks him how many pieces he wishes to eat and then tells him how far he must run to burn up the calories he will consume.

```
]RUN
How many pieces did you eat? 4
You must run 15 miles.

]
```

19. Have the computer evaluate the expression $12x + 7y$ for the following data:

x	3	7	12
y	2	9	–4

20. Write a program that will allow a string A$ to be entered. Then have the computer print the string followed by " KEEPS THE DOCTOR AWAY."

21. Using an INPUT statement, write a program that will compute the volume of a room given its length, width, and height.

22. Write a program to compute the areas (cm^2) of circles with radii 5.0 cm., 3.0 cm. and 8.0 cm. Have the output in the form "AREA OF CIRCLE=" with two spaces between each of the printed lines.

23. Just as three-dimensional objects are measured by volume, so four-dimensional objects are measured by tesseracts. Have your program ask for the dimensions of a four-dimensional object (height, width, length, and "presence") and print the object's tesseract.

24. With the equation $E = MC^2$, Einstein predicted that energy could be produced from matter. If the average human hair weighs a tenth of a gram and the town of Woodsylvania uses 2×10^{11} units of energy in a day, find out how many hairs from Einstein's head would be required to supply the town for a day ($C = 3 \times 10^{10}$).

25. Use the computer to calculate your library fines. Enter the number of books you have borrowed and how many days late they are. Have the computer print the amount of your fine if you are charged 10¢ per day per book.

26. The perimeter of a triangle is equal to the sum of the lengths of the three sides of the triangle. The semiperimeter is one-half of the perimeter. A triangle has sides of lengths 13 cm, 8 cm, 11 cm. A second triangle has sides of 21 ft, 16 ft, 12 ft. Write a program that reads these measurements from a DATA statement and then prints the semiperimeter of each triangle, showing the correct units. The output should look like this:

```
SEMIPERIMETER OF FIRST TRIANGLE 16 CM.

SEMIPERIMETER OF SECOND TRIANGLE 24.5 FT.
```

27. Professional athletes have succeeded in making staggering sums of money through careful negotiations. Of course, the real winner is Uncle Sam, who does not negotiate at all. Write a program which asks for a player's name and salary and then prints the player's name, take-home salary and taxes if the tax rate for his income bracket is 44%

```
]RUN
What is the player's name? JUGGIE RACKSON
What is JUGGIE RACKSON's wage 150000
JUGGIE RACKSON would keep $84000
JUGGIE RACKSON would pay $66000 in taxes.
```

```
]RUN
What is the player's name? BABE RUTH
What is BABE RUTH's wage 42000
BABE RUTH would keep $23520
BABE RUTH would pay $18480 in taxes.
```

28. Sale prices are often deceptive. Write a program to determine the original price of an item, given the sale price and the discount rate.

```
]RUN
Sale price? 3.78
Discount rate? 10
The original price was $4.2

]
```

29. The area of a triangle can be found by multiplying one-half times the length of the base times the length of the altitude (A = .5 * base * height). Write a program that allows the user to enter from the keyboard the base and altitude of a triangle and then skips a line before printing out the area of the triangle.

```
]RUN
What is the base? 10
What is the altitude? 5

The area is 25

]
```

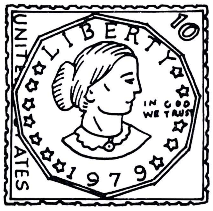
30. A state has a 7 percent sales tax. Write a program that will allow you to INPUT the names and prices (before taxes) of different items found in a department store and then print the item, tax, and the price of the item including tax.

```
]RUN
Item's name? COAT
What is its price? 65
COAT has a tax of $4.55 and costs $69.55
Item's name? TENNIS RACKET
What is its price? 23
TENNIS RACKET has a tax of $1.61 and costs $24.61
Item's name?
```

31. In an election in Grime City candidate Sloth ran against candidate Graft for mayor. Below is a listing of the number of votes each candidate received in each ward. What total vote did each candidate receive? What percentage of the total vote did each candidate receive?

Candidate	Sloth	Graft
Ward 1	528	210
2	313	721
3	1003	822
4	413	1107
5	516	1700

32. A Susan B. Anthony dollar of diameter 2.6 centimeters rests on a square postage stamp as shown. Have the computer find the area in square centimeters of the part of the stamp not covered by the coin.



33. Given the assumption that you sleep a healthy 8 hours a night, have the computer print the number of hours of your life which you have spent sleeping. Input the date of your birth and today's date in numeric form (e.g., 6, 4, 62). Use 365 days in each year and 30 days in a month.

34. Using an INPUT statement, write a program which averages each of the following sets of numbers: (2, 7, 15, 13); (8, 5, 2, 3); (12, 19, 4); (15, 7, 19, 24, 37). Note that the sets do not contain the same number of elements.

DECISIONS AND LOOPS

IF...THEN

AND, OR

END

FOR...TO

STEP

NEXT

RESTORE

The statements presented in Chapter One allow the user to perform routine calculations and to print numbers or strings in different formats. The computer executes routine calculations according to specific rules of precedence called Order of Operations. The higher capabilities of a computer are not called upon until the computer is used to make decisions or to carry out a process many times. This chapter will first outline the Order of Operations and then present the conditional statement IF... THEN, which allows the computer to make simple decisions. Finally this chapter will introduce the statements FOR... TO... STEP, NEXT, which allow loops to be established in a convenient manner.

Order of Operations

What is the value of 2 + 20/4? Should the 2 be added first to the 20 and then the division performed afterward, or should the 20 first be divided by 4 and then be increased by 2? The latter procedure is correct because division is carried out before addition. When parentheses are used, the operations in the innermost parentheses are completed first. For example, (2 + 20)/4 = 22/4 = 5.5.

The computer performs arithmetic operations in the same order as that employed by mathematicians. Quantities in parentheses are evaluated first (starting from the innermost), followed by raising to a power, then by multiplication and division, and finally by addition and subtraction. Operations of equal priority are carried out from left to right. For example:

6/3*2	The result is 4. Although the answer might seem to be 6/6 = 1, the computer starts at the left and performs the division first and then the result of the division is multiplied by 2.
3*(5 + 6)	The result is 33. The computer first adds 5 and 6 and then multiplies the sum by 3.
5 + (3*(6/2))	The result is 14. The computer first divides 6 by 2, the operation within the innermost parentheses. Next it multiplies that result by 3 and finally adds 5.
12 + 4/0	The result is an error message. The computer does not divide by zero
2↑3↑2	The result is 64, just as it would be for (2↑3)↑2. Again, the left-to-right rule is in operation. Remember that ↑ is used to raise to a power.

IF... THEN

The statements introduced in Chapter One (GOTO, PRINT, etc.) are called unconditional because the computer will always execute them. In contrast, the IF... THEN statement is conditional because the action taken depends upon whether the condition is found to be true.

The simplest form of the IF . . . THEN statement is:

IF <condition> THEN <line number>

The 'condition' portion of the statement compares two quantities which must be separated by one of the symbols indicated in the table below:

SYMBOL	MEANING
=	equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

An example of an IF . . . THEN statement is:

20 IF X > 5 THEN 60

When the condition in line 20 (X is greater than 5) is true, the computer jumps to line 60. When the condition is false, as when X is less than or equal to 5, the computer proceeds to the next line of the program.

Program 2.1

This program determines whether the value entered for X is the solution to the equation 2X − 18 = 0. The computer decides which of two messages is to be printed:

```
10   INPUT X
20   IF 2 * X - 18 = 0 THEN 50
30   PRINT X;" IS NOT THE SOLUTION"
40   GOTO 10
50   PRINT X;" IS THE SOLUTION"

]RUN
?15
15 IS NOT THE SOLUTION
?4
4 IS NOT THE SOLUTION
?9
9 IS THE SOLUTION

]
```

Note the necessity of line 40. Without it, the program would print "IS THE SOLUTION" even when the solution was not X. The line number sequence that the computer follows when X is the solution is 10, 20, 50; and when X is not the solution, it follows the line number sequence 10, 20, 30, 40, 10.

The IF . . . THEN statement can be used to compare two strings. Here the symbols greater than (>), less than (<), equal to (=), etc., now refer to an alphabetical rather than a numerical order.

Program 2.2

The following program determines if A$ is alphabetically before, the same, or after B$:

```
10   PRINT
20   INPUT "ENTER TWO STRINGS? ";A$,B$
30   IF A$ = B$ THEN 70
40   IF A$ > B$ THEN 90
50   PRINT A$;" IS BEFORE ";B$
60   GOTO 10
70   PRINT A$;" IS EQUIVALENT TO ";B$
80   GOTO 10
90   PRINT A$;" IS AFTER ";B$
100   GOTO 10

]RUN

ENTER TWO STRINGS? MCINTOSH,DELICIOUS
MCINTOSH IS AFTER DELICIOUS

ENTER TWO STRINGS? A,C
A IS BEFORE C

ENTER TWO STRINGS? SYSTEM,SYSTEM
SYSTEM IS EQUIVALENT TO SYSTEM

ENTER TWO STRINGS?

BREAK IN 20
]
```

Line 30 directs the computer to line 70 if A$ and B$ are equivalent. Line 40 causes a jump to line 90 if A$ comes after B$ in the alphabet. Line 50 is reached only if both conditions in lines 30 and 40 fail and, therefore, A$ comes before B$ in the alphabet.

Extended Use of the IF . . . THEN Statement, Multiple Statement Lines

In the preceding section the simplest form of the IF . . . THEN statement was given as:

IF <condition> THEN <line number>

More flexible usage of IF . . . THEN permits a statement instead of a line number to follow the word THEN. This means that the statement after the condition will be executed only if the condition is true. The form is:

IF <condition> THEN <statement>

The following examples illustrate legal use of the IF . . . THEN statement:

IF X <> 0 THEN PRINT X
IF Z>1 THEN PRINT "IT IS LARGER THAN ONE"
IF R< = 0 THEN R = X + 2

The computer allows a series of statements to be entered on a single program line if they are separated from each other by colons (:). Multiple statement lines have special meaning when used with IF . . . THEN. *All* statements following THEN will be executed when the condition following IF is true. *No* statements following THEN will be executed when the condition is false. For example,

60 IF X = 5 THEN PRINT "GOOD": GOTO 80

has two statements on one line. If the condition X = 5 is true, then "GOOD" will be printed and program control will be passed to line 80. If the condition is false, neither of these actions is taken. Instead, the program will continue on to the line following line 60. This means either the PRINT "GOOD" and the GOTO 80 statements are executed, or neither is executed. Whichever of the two occurs depends upon whether the condition X = 5 is true.

Program 2.3

Program 2.1 can be shortened by using multiple statement techniques:

```
10  INPUT X
20  IF 2 * X - 18 = 0 THEN  PRINT X;" IS THE SOLUTION": GOTO 10
30  PRINT X;" IS NOT THE SOLUTION": GOTO 10

]RUN
?15
15 IS NOT THE SOLUTION
?2
2 IS NOT THE SOLUTION
?9
9 IS THE SOLUTION
?

BREAK IN 10
]
```

Note that the statement GOTO 10 in line 20 is executed *only* if the condition in the IF . . . THEN statement is true. Otherwise the program proceeds to line 30. The statement GOTO 10 in line 30 will always be executed if line 30 is reached since it is not preceded by an IF . . . THEN statement on the same line.

Review

1. Allow two numbers, A and B, to be entered in the computer and then print the two numbers in descending order.
2. Allow two names to be entered and then printed in alphabetical order.

```
]RUN
ENTER TWO LAST NAMES. PATRICK,GREER
GREER
PATRICK
ENTER TWO LAST NAMES.

BREAK IN 10
]
```

AND, OR

The statement modifiers AND and OR can be used in the IF . . . THEN statement when more than one condition is to be considered. The statement

20 IF X > 5 AND Y = 3 THEN 50

will cause a transfer to line 50 only if *both* conditions are true. On the other hand, the statement

20 IF X > 5 OR Y = 3 THEN 50

will cause program control to be transferred to line 50 if *either or both* conditions are true.

Program 2.4

The following program uses the OR modifier and reviews much of what has been presented:

```
10   INPUT "WHO ARE YOU? ";N$
20   IF N$ = "JAMES BOND" OR N$ = "007" THEN 50
30   PRINT "THIS MISSION IS NOT MEANT FOR YOU!"
40   PRINT : GOTO 10
50   PRINT "THESE ORDERS ARE FOR YOUR EYES ONLY!"
60   INPUT "ARE YOU ALONE? ";A$
70   IF A$ <  > "YES" THEN  PRINT "TRY LATER.": GOTO 10
80   PRINT : PRINT "DR. ZHIVAGO HAS ESCAPED! YOUR"
90   PRINT "MISSION SHOULD YOU DECIDE TO ACCEPT"
100   PRINT "IT, IS TO SEEK OUT ZHIVAGO AND"
110   PRINT "RETURN HIM TO THE SPANDAU PRISON."
```

```
]RUN
WHO ARE YOU? BRAVE SIR ROBIN
THIS MISSION IS NOT MEANT FOR YOU!

WHO ARE YOU? JAMES BOND
THESE ORDERS ARE FOR YOUR EYES ONLY!
ARE YOU ALONE? NO
TRY LATER.
WHO ARE YOU? 007
THESE ORDERS ARE FOR YOUR EYES ONLY!
ARE YOU ALONE? YES

DR. ZHIVAGO HAS ESCAPED! YOUR
MISSION SHOULD YOU DECIDE TO ACCEPT
IT, IS TO SEEK OUT ZHIVAGO AND
RETURN HIM TO THE SPANDAU PRISON.

]
```

Line 10 asks the user for a name. The name entered as N$ is compared with the two names at line 20. If N$ is either 'JAMES BOND' or '007', then the program skips to line 50. Otherwise, a message is printed by line 30 and the program returns to line 10. At line 50 a warning message is printed whereas line 60 inquires whether Mr. Bond is alone. Note that in line 70 any response other than 'YES' will cause the program to print 'TRY LATER' and jump back to line 10. Otherwise, the mission description is printed by lines 80 through 110.

END

The END statement is used to terminate the run of a program. For example,

80 END

will cause the run of a program to stop when line 80 is reached. It is possible to place END statements at more than one location within a program, including insertion in an IF . . . THEN statement.

Program 2.5

This program uses the END statement to terminate program execution when the name KERMIT is input:

```
10  INPUT "WHAT IS YOUR NAME? ";N$
20  IF N$ = "KERMIT" THEN  END
30  PRINT "IT'S GOOD TO MEET YOU, ";N$
40  PRINT : GOTO 10

]RUN
WHAT IS YOUR NAME? CYNTHIA
IT'S GOOD TO MEET YOU, CYNTHIA

WHAT IS YOUR NAME? KERMIT

]
```

3. Write a program that will allow a number N to be entered. If the number is between 25 and 112, then the computer should indicate this; otherwise, the computer should say that the number is out of range.

```
]RUN
ENTER A NUMBER. 14
14 IS OUT OF RANGE
ENTER A NUMBER. 54
54 IS BETWEEN 25 AND 112
ENTER A NUMBER. 112
112 IS OUT OF RANGE
ENTER A NUMBER.

BREAK IN 10
```

4. Write a program in which the user enters a string N$. If it comes alphabetically before Garbage or after Trash, then have the computer print "YES"; otherwise have it print "NO".

FOR . . . TO . . . STEP, NEXT

The FOR . . . TO . . . STEP, NEXT statements provide a simple way of establishing a loop. A loop is a section of a program designed to be executed repeatedly. The FOR . . . TO . . . STEP, NEXT loops provide a method for generating a large sequence of numbers in a case where each number in the sequence differs from its predecessor by a constant amount. The general form of the FOR . . . TO . . . STEP, NEXT statement is:

```
┌──────FOR <variable> = <starting value> TO <ending value> STEP <increment>
│          •
│          •
│          •
└──────NEXT <variable>
```

Note that the 'variable' after FOR and NEXT must be the same. A string variable cannot be used in a FOR . . . NEXT loop. The STEP portion may be omitted if the increment equals +1.

For example:

```
┌──────10 FOR X = 2 TO 6
│          •
│          •
│          •
└──────40 NEXT X
```

```
70 FOR H1 = N*2 TO 26 STEP +2
        •
        •
        •
100 NEXT H1
```

```
120 FOR T = 10 TO 0 STEP -1
        •
        •
        •
180 NEXT T
```

Note that a loop is formed by the FOR... TO... STEP and NEXT statements. In the example,

```
30 FOR N = 3 TO 11 STEP 2
        •
        •
        •
80 NEXT N
```

the variable N starts at line 30 with a value of 3. N retains the value of 3 until the NEXT N statement is encountered at line 80. At this point N is increased by the STEP value. In this example N changes from 3 to 5 to 7, etc. All the statements in the lines occurring between lines 30 and 80 are executed in sequence during each consecutive pass through the loop. The program continues to return from line 80 to the line immediately following line 30 until N exceeds the specified limit of 11. At this point the program exits from the loop and moves on to the line following 80.

Program 2.6

This program uses READ, DATA, whereas Program 2.7 uses a FOR... TO... STEP, NEXT loop to print all integers between 10 and 30, inclusive.

```
10   READ N
20   PRINT N;"   ";
30   GOTO 10
40   PRINT " DONE"
50   DATA  10,11,12,13,14,15,16,17,18
60   DATA  19,20,21,22,23,24,25,26,27
70   DATA  28,29,30

]RUN
10  11  12  13  14  15  16  17  18  19
20  21  22  23  24  25  26  27  28  29
30
?OUT OF DATA ERROR IN 10
]
```

Program 2.7

```
10   FOR N = 10 TO 30
20   PRINT N;"   ";
30   NEXT N
40   PRINT " DONE"

]RUN
10   11   12   13   14   15   16   17   18   19
20   21   22   23   24   25   26   27   28   29
30    DONE

]
```

Program 2.6 continues to read data until all of it is exhausted. Program 2.7 generates all of the values of N between 10 and 30. Note that the loop in Program 2.7 is completed by a NEXT statement, not a GOTO statement. Therefore, unlike Program 2.6, Program 2.7 can proceed to line 40.

Program 2.8

This program finds solutions to the compound condition $5X + 3 < 100$ and $2X^2 - 1 > 50$ and tests all odd integers from 1 to 25:

```
10   FOR X = 1 TO 25 STEP 2
20   IF 5 * X + 3 < 100 AND 2 * X ^ 2 - 1 > 50 THEN  PRINT X;" ";
30   NEXT X
40   PRINT " DONE"
50   END

]RUN
7 9 11 13 15 17 19  DONE

]
```

Note that the symbol (∧) is used to denote ‘raised to the power’ (e.g., X∧2 means X∗X). Lines 10 and 30 create a loop for testing the conditions located at line 20. The loop starts at line 10 with X = 1. Each time the program reaches the NEXT X statement at line 30, X is incremented by the STEP value, which is 2. The program will then return to line 20 unless the value of the loop variable X has exceeded its maximum value of 25, at which point the loop will be exited. Remember that a FOR loop is not completed by a GOTO statement but by a NEXT statement.

Program 2.9

The following program illustrates the use of a negative STEP value. The integers from 1 to 10 are printed in descending order:

```
10   FOR X = 10 TO 1 STEP   - 1
20   PRINT X;" ";
30   NEXT X
40   PRINT
50   END

]RUN
10 9 8 7 6 5 4 3 2 1

]
```

Review

5. Write a program that prints the integers between 1 and 25, inclusive.

6. Using a FOR . . . TO . . . STEP, NEXT loop, have the computer print the following:

```
]RUN
20   18   16   14   12   10

]
```

7. Write a program that will allow a number N to be entered. Using N as a STEP value, print numbers between 8 and 20.

```
]RUN
STEP VALUE? 2
8 10 12 14 16 18 20
]
```

RESTORE

At times the user might find it desirable to employ a set of data more than once. The RESTORE statement makes it possible to read data again starting at the beginning of the data. For example,

40 RESTORE

causes the next READ statement encountered to read data starting with the first item in the first DATA statement.

Program 2.10

This program searches DATA statements for a specific student and then prints the student's grades. It employs the RESTORE statement so that each time a student's name is entered a searching process starts at the beginning of the first DATA statement. How would the program run if line 30 were deleted?

```
10  PRINT
20  INPUT "NAME? ";B$
30  RESTORE
40  FOR P = 1 TO 6
50  READ N$,A,B,C,D
60  IF N$ = B$ THEN 90
70  NEXT P
80  PRINT "STUDENT ";B$;" NOT FOUND": GOTO 10
90  PRINT N$;" HAS THE FOLLOWING GRADES:"
100  PRINT A;"  ";B;"  ";C;"  ";D
110  GOTO 10
200  DATA  WATERS,83,75,92,80,TRANTER,74,81,92,76
210  DATA  SAVAGE,72,81,63,60,RICKARD,99,84,92,87
220  DATA  MULRYAN,100,93,82,89,LEMONE,94,93,85,94
230  END

]RUN

NAME? RICKARD
RICKARD HAS THE FOLLOWING GRADES:
99  84  92  87

NAME? LEMONE
LEMONE HAS THE FOLLOWING GRADES:
94  93  85  94

NAME? BROWNE
STUDENT BROWNE NOT FOUND

NAME? MULRYAN
MULRYAN HAS THE FOLLOWING GRADES:
100  93  82  89

NAME? SAVAGE
SAVAGE HAS THE FOLLOWING GRADES:
72  81  63  60

NAME? WATERS
WATERS HAS THE FOLLOWING GRADES:
83  75  92  80

NAME?

BREAK IN 20
]
```

Program 2.11

This program incorporates the major topics covered in this chapter, which the reader should now be familiar with, and demonstrates how they might be used together in a single program.

```
10   INPUT "WORD TO SKIP? ";S$
20   FOR W = 1 TO 10
30   READ W$
40   IF W$ = S$ THEN 60
50   PRINT W$
60   NEXT W
70   PRINT
80   INPUT "REPEAT? ";R$
90   IF R$ <   > "YES" THEN   END
100   RESTORE
110   PRINT
120   GOTO 10
130   DATA   THIS,IS,A,SAMPLE,OF,THE
140   DATA   MATERIAL,IN,CHAPTER,TWO

]RUN
WORD TO SKIP? CHAPTER
THIS
IS
A
SAMPLE
OF
THE
MATERIAL
IN
TWO

REPEAT? YES

WORD TO SKIP? THE
THIS
IS
A
SAMPLE
OF
MATERIAL
IN
CHAPTER
TWO

REPEAT? NO

]
```

Exercises

1. Write a program to allow two numbers (A and B) to be entered. Have the computer compare them and print a message stating whether A is less than, equal to, or greater than B.

2. Write a program which allows three names to be entered as A$, B$ and C$. Have the computer print the name which is alphabetically last.

3. Perform each of the following computations on paper. Check your results by using immediate mode on the computer.

   (A) 3↑2↑3
   (B) 5 - 4↑2
   (C) 3*(5 + 16)
   (D) 5 + 3 * 6/2
   (E) 640/10/2 * 5
   (F) 5 + 3 * 4 - 1
   (G) 2↑3↑2
   (H) 2↑(3↑2)
   (I) 64/4 * 0.5 + ((1 + 5) * 2↑3) * 1/(2 * 4)

4. Write a program which prints six exclamation marks if BIGWOW is entered for the string X$ but which otherwise prints six question marks. The program is run until interrupted by the user.

5. Use only one line number to enter two strings (A$ and B$) and to print output similar to the following:

```
]RUN
?START,FINISH
START               FINISH
FINISH              START

]
```

6. Allow a number (X) to be entered. Print the message "NOT BETWEEN" if X is either less than -24 or greater than 17. Only one IF . . . THEN statement is permitted.

7. Allow a string (A$) to be entered. Print the message "A$ IS BETWEEN" If A$ comes between "DOWN" and "UP" alphabetically. Only one IF . . . THEN statement is permitted.

8. Allow a number (X) to be entered. Print "IN THE INTERVAL" if X satisfies the inequality 25 < X < 75, but otherwise print "NOT IN THE INTERVAL." Only one AND and one IF . . . THEN statement are to be used.

9. Rewrite the program of the preceding exercise, using OR instead of AND.

10. Use only one PRINT statement to produce the following rectangle:

```
]RUN
************
************
************

]
```

11. Print the cubes of the odd integers from 11 to -11, inclusive, in descending order.

12. Print all the integers which end in 4 from 4 to 84, inclusive.

13. Use a loop to print a horizontal line of 20 asterisks.

14. Write a one line Immediate Mode command for each of the following:
    (A) Find the value of 143 x 74.
    (B) Find the average of 53, 72, 81 and 76.
    (C) Find which is larger, X ↑ Y or Y ↑ X, when X = 4 and Y = 5.

15. Print all of the integers in the set 10, 13, 16, 19, . . . , 94, 97.

16. Using a loop, have the computer print a letter I as follows:

```
]RUN
*****
 ***
 ***
 ***
 ***
 ***
 ***
*****

]
```

17. Below is a list of various creatures and the weapon necessary to destroy each:

Creature	**Weapon**
Lich	Fire Ball
Mummy	Flaming Torch
Werewolf	Silver Bullet
Vampire	Wooden Stake
Medusa	Sharp Sword
Triffid	Fire Hose

Using READ and DATA, have the computer state what weapon is to be used to destroy a given creature.

18. Marvelous Matilda has written the following program using too many GOTO statements and cannot figure out in what order the lines of the program will be executed. Assist her by listing the line numbers in the sequence in which the computer will execute them:

```
10  READ A,B
20  IF A > 4 OR B > 4 THEN 50
30  IF A < 1 AND B < 1 THEN 60
40  GOTO 70
50 X = A + B: GOTO 80
60 X = A * B: GOTO 80
70 X = A / B
80  IF X > 1 THEN  PRINT X: GOTO 10
90  DATA  5,3,-1,-2,0,2
```

19. John Doe's brother Jim has been assigned the following program for homework. Jim is not in shape today, so assist him by stating the exact order in which the lines of the following program are to be executed:

```
10  READ A,B,C
20 S = A + B * C
30  IF S = 10 THEN  RESTORE : READ S
40  PRINT S,
50  IF S = 14 THEN  END
60  GOTO 10
70  DATA  4,2,3,6,0,2,7
```

20. Write a program which asks for a person's age. If the person is 16 years or older, have the computer print "YOU ARE OLD ENOUGH TO DRIVE A CAR." Otherwise, have the computer indicate how many years the person must wait before being able to drive.

```
]RUN
How old are you? 16
You are old enough to drive a car.

]
```

21. The Happy Holiday Motel has 10 rooms. Have the computer print a label for each room's door indicating the room number. For example:

```
]RUN

------------------------------
   Happy Holiday Motel
         Room 1
------------------------------
```

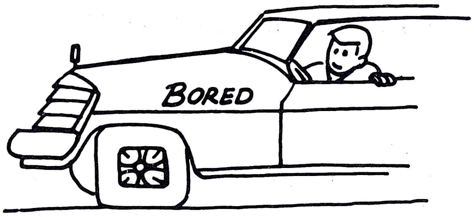
22. As candidate for mayor, you are very busy. Use the computer to print thank you letters to 10 people who have contributed money to your election campaign. Be sure to mention the exact amount each person has contributed.

```
]RUN

Dear Rich Bryburry,
   Thank you for your generous contribution
of 25000 dollars to my election campaign. Maybe
next year we will have better luck!
                              Sincerely,

                              Smiley R. Politico
```

23. Write a program which will produce the following table:

```
]RUN
X                X^2              X^3

2                4                8
4                16               64
6                36               216
8                64               512
10               100              1000

]
```

24. The Bored Auto Company has done it again! Some models of their cars may be difficult to drive because their wheels are not exactly round. Cars with model numbers 102, 780, 119, 229, 189, and 195 have been found to have a defect. Write a computer program that allows 10 of their customers to enter the model number of their car to find out whether or not it is defective.



25. Using only two print statements, write a program to print a triangle that in N lines high and N columns wide. For example:

```
]RUN
?5
*
**
***
****
*****

]
```

26. Write a program that will produce the following triangle. The figure is centered on TAB(10):

```
]RUN
         *
        * *
       *   *
      *     *
     *       *
    *         *
   *************

]
```

27. The following table contains employee performance data for the Tippecanoe Typing Company:

Employee	**Performance**
Oakley	69%
Howe	92%
Anderson	96%
Wolley	88%
Goerz	74%

Tippecanoe Typing is suffering from financial difficulties and needs to cut back on its staff. Using READ, DATA and a loop, have the computer print notices of dismissal for any employee whose production performance is below 75%.

```
]RUN

Dear Oakley,
 I am sorry that I must fire you.
You have been such a fine employee
with a performance rating of 69%
I'm sure you'll have no trouble
finding another job.
                   Sincerely,

                   George Shwabb


Dear Goerz,
 I am sorry that I must fire you.
You have been such a fine employee
with a performance rating of 74%
I'm sure you'll have no trouble
finding another job.
                   Sincerely,

                   George Shwabb


]
```

28. Wayne Peber bought stock two years ago and wants to use the computer to calculate his profit or loss. He bought 200 shares of Consolidated Technologies at $85.58 per share and 400 shares of American Amalgamated Securities at $35.60 per share. Today C.T. is worth $70.82 a share and A.A.S. is worth $47.32 a share. What is his profit or loss?

29. The Exploitation Oil Company uses the computer to determine the weekly wages of its employees by inputting the hours worked and the hourly wage for each employee. If the employee works over 40 hours, he or she is paid one and a half times the hourly rate for each additional hour.

```
]RUN
Hours worked? 45
Hourly wage? 10.00
The wage for the week is $475

]
```

30. The Last Chance Finance Company is charging a rate of 2% per month on all loans it is making. Write a program that allows Last Chance to calculate the monthly payments charged a customer using the following formula:

$$\text{Monthly Payment} = \frac{L*I}{1-(1+I)^{-m}}$$

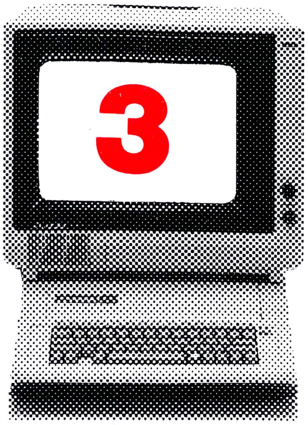
where L = Amount loaned
I = Interest (monthly)
m = Number of months to be loaned

```
]RUN
The amount of the loan? 100
Length of the loan in years? 5
The monthly payment is $2.87679654
The total amount paid will be $172.607792

]
```

31. You have $200.00 to spend on a buying spree. Write a program that, as you buy merchandise, subtracts the cost and the appropriate sales tax (5%) from your remaining money and shows your present total. The program should prevent you from buying items that cost more than you have.

```
]RUN
How much does the item cost? 10.00
Your total is now $189.5

How much does the item cost? 250.00
You don't have enough money.
How much does the item cost? 25.00
Your total is now $163.25

How much does the item cost? 0

]
```

32. Have the computer find all odd integers from 5 to 25 which are simultaneous solutions of the inequalities $X^3 > 500$ and $X^2 + 3X + 2 < 700$. Print only the solutions.



COMPUTER GAMES

RND

INT

POS

HTAB

VTAB

HOME

FLASH

INVERSE

NORMAL

GET

This chapter will introduce material which should be interesting, practical and entertaining. In fact, the emphasis will be placed on techniques to involve the user in computer games, which are often the most enjoyable way to learn computer programming.

RND

In many computer simulations that involve science problems and also in a variety of computer games, the process of generating random numbers is essential. The RND function generates random numbers on the computer. The statement

$$X = RND(1)$$

will assign X a random number such that $0 < X < 1$.

Program 3.1

This program will print 4 random numbers:

```
10   FOR I = 1 TO 4
20   PRINT  RND (1),
30   NEXT I

]RUN
.846447204        .841536558
.591965711        .26800113

]
```

INT

The statement A = INT(X) sets A to the largest integer that is not greater than X. This function does not round off a positive number but simply truncates its decimals. In the case of a negative number, it takes the next lower negative integer. For example:

$$\text{INT }(3.7640) = 3 \qquad \text{INT }(5.9) = 5$$

$$\text{INT }(-1.7) = -2 \qquad \text{INT }(-3.01) = -4$$

At times it is preferable to have the computer pick random integers instead of long random decimals. This can be accomplished by using INT. The following formula can be used to assign X a random integer between A and B, inclusive:

$$X = \text{INT }((B - A + 1)*\text{RND}(1) + A)$$

For example, the statement

$$X = \mathrm{INT}(26*\mathrm{RND}(1) + 75)$$

will assign X a random integer between 75 and 100, inclusive.

Program 3.2

This game program selects a random number from 1 to 100 and then gives the player an unlimited number of chances to guess it. After each guess the computer informs the player that the guess is too high, too low, or correct.

```
10  PRINT "I'M THINKING OF A NUMBER FROM 1 TO 100."
20 R =  INT (100 *  RND (1) + 1)
30  INPUT "WHAT IS YOUR GUESS? ";G
40  IF G < R THEN  PRINT "TOO LOW!": GOTO 30
50  IF G > R THEN  PRINT "TOO HIGH!": GOTO 30
60  PRINT "THAT IS CORRECT!!!"

]RUN
I'M THINKING OF A NUMBER FROM 1 TO 100.
WHAT IS YOUR GUESS? 50
TOO LOW!
WHAT IS YOUR GUESS? 75
TOO LOW!
WHAT IS YOUR GUESS? 86
TOO LOW!
WHAT IS YOUR GUESS? 90
THAT IS CORRECT!!!

]
```

Review

1. Write a program that will generate 2 random integers between 50 and 150 and find their product.

```
]RUN
92 MULTIPLIED BY 138 IS 12696

]
```

2. Write a program similar to Program 3.2 which picks a random number between 1 and 50, inclusive, and gives the player only five attempts to guess it.

```
]RUN
I'M THINKING OF A NUMBER FROM 1 TO 50.
WHAT IS YOUR GUESS? 25
TOO LOW!
WHAT IS YOUR GUESS? 37
TOO LOW!
WHAT IS YOUR GUESS? 44
TOO HIGH!
WHAT IS YOUR GUESS? 40
TOO LOW!
WHAT IS YOUR GUESS? 41
TOO LOW!
YOU'VE HAD FIVE GUESSES NOW.
THE NUMBER WAS 42

]
```

Summation

If the programmer decides to keep score for the number guessing game (Program 3.2), some technique will have to be devised to keep count of the number of guesses taken. One possible technique is to use a summation statement of the form:

```
30 A = A + 1
```

The statement A = A + 1 makes no sense in mathematics since A can equal A but not A + 1. The computer, however, interprets the equal sign to mean "is replaced by" rather than "equal to", and each time it encounters line 30 above, it takes the present value of A, adds 1 to it, and makes that sum the new value of A.

Program 3.3

This program demonstrates how the summation statement works by printing the values of the variable A until the program is halted with CONTROL-C.

```
10 A = A + 1
20  PRINT A;"   ";
30  GOTO 10

]RUN
1  2  3  4  5  6  7  8  9  10  11  12
13  14  15  16  17  18

BREAK IN 20
]
```

If line 10 is replaced with

```
10 A = A + 5
```

then 5 will be added to A each time the statement is encountered.

```
]10 A = A + 5

]RUN
5   10   15   20   25   30   35   40   45   50
55   60   65   70   75   80

BREAK IN 20
]
```

In this problem the initial value of A is zero for both runs because the computer sets any undefined numeric variables to zero at the start of a run.

Program 3.4

Here, Program 3.2 has been modified to record the number of turns required to guess the random number. In this case the variable A acts as a counter, increasing in value by 1 after each guess:

```
10  PRINT "I'M THINKING OF A NUMBER FROM 1 TO 100."
20 R =  INT (100 *  RND (1) + 1)
30  INPUT "WHAT IS YOUR GUESS? ";G
35 A = A + 1
40  IF G < R THEN  PRINT "TOO LOW!": GOTO 30
50  IF G > R THEN  PRINT "TOO HIGH!": GOTO 30
60  PRINT "THAT IS CORRECT!!!"
70  PRINT "THAT TOOK YOU ";A;" GUESSES."

]RUN
I'M THINKING OF A NUMBER FROM 1 TO 100.
WHAT IS YOUR GUESS? 50
TOO LOW!
WHAT IS YOUR GUESS? 75
TOO HIGH!
WHAT IS YOUR GUESS? 62
TOO HIGH!
WHAT IS YOUR GUESS? 56
THAT IS CORRECT!!!
THAT TOOK YOU 4 GUESSES.

]
```

Rounding Errors

Because the computer has a finite capacity, any numerical computations involving infinitely repeating decimals cannot be accurate (for example, 1/3 = .3333 . . ., which the computer truncates to a limited number of digits). Since the computer uses binary numbers (0 and 1), any fraction whose denominator is not an integral power of 2 (2, 4, 8, 16, etc.) will be an infinitely repeating decimal and therefore will be truncated by the computer. In Chapter Eight binary numbers are examined in greater detail.

Program 3.5

This program illustrates the processes of summation and rounding error:

```
10  FOR X = 1 TO 1500
20 A = A + (1 / 2)
30 B = B + (1 / 3)
40  NEXT X
50  PRINT "A = ";A
60  PRINT "B = ";B

]RUN
A = 750
B = 500.000023

]
```

The final value of A is the result of adding ½ 1500 times, which comes out to exactly 750. B, however, which is the result of adding ⅓ 1500 times, does not come out to exactly 500 because of the rounding error. At times the computer does not recognize an equality when one in fact exists. If the line

```
70 IF B = 500 THEN PRINT ‘‘EQUAL’’
```

were added to the above program, then the computer would not print the word “EQUAL.”

Review

3. Select 50 random integers between 0 and 9, inclusive, and have the computer tell how many of the numbers were from 0-4 and how many were from 5-9.

```
]RUN
THERE WERE 27 NUMBERS FROM 0 TO 4.
THERE WERE 23 NUMBERS FROM 5 TO 9.

]
```

POS

The POS function is used to find the current horizontal position of the cursor. The statement

```
X = POS(0)
```

will assign X a value corresponding to the horizontal position (the current TAB position) of the cursor on the screen. On the Apple, X = 1 for the left edge and 40 for the right edge. If 80 column mode is in effect X = 80 for the right edge.

HTAB, VTAB, and HOME

HTAB and VTAB are used without PRINT statements to move the cursor around the screen. VTAB moves the cursor vertically on the screen. VTAB 1 refers to the top of the screen, and VTAB 24 moves the cursor to the bottom of the screen. Unlike TAB, HTAB can move the cursor to the left as well as the right. HTAB 1 will place the cursor at the left edge of the screen, and HTAB 40 will place the cursor at the right edge of the screen. HOME will erase the screen and place the cursor in the upper left-hand corner of the screen.

Program 3.6

This program will randomly place 20 asterisks or "stars" on the video screen.

```
5  HOME
10  FOR N = 1 TO 20
20 X =  INT (40 *  RND (1) + 1)
30 Y =  INT (24 *  RND (1) + 1)
40  HTAB X
50  VTAB Y
60  PRINT "*";
70  NEXT N


]RUN


                      *
                                *

                  *  *
*                        *
 *  *                            *
                                 *
                          *



                                    *

*                    *
              *


                       *

                         **
                           *  *

]
```

FLASH, INVERSE, and NORMAL

The output by a PRINT statement can have one of three characteristics: the output can blink, it can be in inverse, or it can be normal. The FLASH statement causes the output of subsequent PRINT statements to blink on the screen. That is, everything printed on the screen after the FLASH statement is executed will blink. The INVERSE statement allows characters to be printed in inverse, much like a photographic negative. Normally, characters appear on the screen in white on a black background. However, with INVERSE, characters can appear in black on a white background, but the white background appears only behind those characters printed. The NORMAL statement cancels the effect that FLASH and INVERSE have on the PRINT statement. When INVERSE or FLASH is used on the Apple //e computer, lower case characters (e.g., a-z) will produce unusual results. For example, typing a lower case T in FLASH mode will produce a blinking digit 4.

Program 3.7

This program illustrates the effect of the FLASH, INVERSE, and NORMAL statements.

```
10  PRINT "SUNNY LOVES ALICE"
20  FLASH
30  PRINT "ALEX WENT TO EUROPE"
40  INVERSE
50  PRINT "THIS IS REVERSED"
60  NORMAL
70  PRINT "ADAM IS MY NEPHEW"

]RUN
SUNNY LOVES ALICE
ALEX WENT TO EUROPE
THIS IS REVERSED
ADAM IS MY NEPHEW

]
```

Note that only one mode can be active at a time. The INVERSE statement at line 40 overides the FLASH statement at line 20. Thus, the only text that flashes is that produced by line 30.

Program 3.8

This program is a very simple version of STAR WARS. The computer randomly places a "star" on the screen and asks the player for its X and Y coordinates. If the player chooses the wrong coordinates, the computer prints the message MISSED and indicates whether each coordinate guessed is too high or too low. When the star is hit, the message HIT is flashed. The player starts with a score of 100 points and loses 10 points for each miss. The score is given at the end of the game.

Type the program into the Apple, and try to play it. By expanding and refining the program, you can make it considerably more sophisticated and intricate.

```
10 A = 100
20  HOME
30 X =  INT (40 *  RND (1) + 1)
40 Y =  INT (20 *  RND (1) + 1)
50  HTAB X
60  VTAB Y
70  PRINT "*"
80  HTAB 1: VTAB 22
90  INPUT "COORDINATES TO FIRE AT (X,Y)? ";X1,Y1
100  VTAB 22
110  IF X = X1 AND Y = Y1 THEN 220
120  IF X1 > X THEN  PRINT "X-COORDINATE IS TOO HIGH.           "
130  IF X1 < X THEN  PRINT "X-COORDINATE IS TOO LOW.            "
140  IF Y1 > Y THEN  PRINT "Y-COORDINATE IS TOO HIGH.           "
150  IF Y1 < Y THEN  PRINT "Y-COORDINATE IS TOO LOW.            "
160 A = A - 10
170  VTAB 21: INVERSE : PRINT "MISSED."
180  FOR D = 1 TO 1000: NEXT D
190  NORMAL
200  VTAB 23: PRINT "                        "
210  GOTO 80
220  VTAB 21: FLASH : PRINT "  HIT!  "
230  NORMAL
240  PRINT "YOUR SCORE IS ";A;"                   "
```

Line 10 sets the initial score to 100 and the screen is cleared by line 20. Lines 30 and 40 pick the random position for the asterisk with a range of values from (1,1) to (40,20). The asterisk is positioned by lines 50 and 60 and then printed by line 70. Next, lines 80 and 90 position a prompt at the lower left of the screen which asks the user for the coordinates to fire at. If either the X value or the Y value is incorrect lines 120 through 150 inform the player whether the input coordinates are too high or too low. Line 160 subtracts 10 points from the current score, and line 170 uses INVERSE mode to inform the player of his or her miss. The delay loop at line 180 is used to pause the computer before the old information is erased by line 200 and the player is asked for the coordinates to fire at next (lines 80-90). The process is repeated until the player guesses the correct coordinates, at which point the conditional at line 110 will cause the computer to jump to line 220. This line uses FLASH to cause the message "HIT" to blink, and the final score is then given by line 240.

GET

The GET statement is used to input a single digit or character from the keyboard. Its format is:

GET <variable>

Unlike the INPUT statement, the character struck is not printed and it is not followed by striking the RETURN key. When a GET statement is executed, the computer waits until a key is struck on the keyboard and assigns the variable that value. For example,

```
100 GET A$
```

assigns A$ a single character which was typed on the keyboard.

Program 3.9

This program uses GET to input a string which may contain commas or any other special character until a minus sign (−) is typed.

```
10  PRINT "INPUT A SENTENCE."
20  PRINT
30  GET A$
40  IF A$ = "-" THEN 80: REM  CHECK FOR END OF INPUT
50  PRINT A$;
60 B$ = B$ + A$
70  GOTO 30
80  PRINT : PRINT
90  PRINT "B$ = ";B$

]RUN
INPUT A SENTENCE.

WHEN I OPENED THE BOX, I FOUND A SURPRISE.

B$ = WHEN I OPENED THE BOX, I FOUND A SURPRISE.

]
```

The GET statement at line 30 assigns A$ a single character struck on the keyboard. Since this value is not automatically printed by the computer it is necessary to place line 50 in the program in order to see what was typed. Line 60 uses a technique similar to summation of numeric variables. When addition is performed on string variables the value assigned is the values of the string variables appended to one another. Thus, at line 60, B$ becomes A$ appended onto the old value of B$. Line 40 causes this process to repeat until a minus sign (−) is typed.

Game Programs

Two game programs are presented here which the programmer should study carefully and then attempt to modify to write other game programs.

Program 3.10

In the game of NIM, two players in turn take matches from a pile originally containing twenty matches. Each player may take either 1 or 2 at a time. The player who takes the last match wins the game. The computer's strategy is simple. If the computer goes first, it will take 2 matches. If the player is allowed to go first, then the computer will match the player's first move. Thereafter, in either case, the computer will select a number of matches so that the total of the player's previous selection and the computer's is 3. For example, if the player picks 2, the computer will pick 1. The following program plays NIM with the user, randomly deciding who goes first.

```
1   REM  M = number of matches
2   REM  C = computer's selection
3   REM  P = player's selection
4   REM
10   PRINT " HERE I HAVE TWENTY MATCHES."
20   PRINT "EACH PLAYER MAY TAKE EITHER"
30   PRINT "1 OR 2 AT A TIME.  THE PLAYER"
40   PRINT "WHO TAKES THE LAST MATCH WINS."
50 M = 20: REM  start with 20 matches
60   IF  RND (1) < .5 THEN  PRINT "YOU GO FIRST.": GOTO 130
70   PRINT "I'LL GO FIRST."
80 C = 2: GOTO 110: REM  computer goes first
90   IF M > 17 THEN C = P: GOTO 110
100 C = 3 - P: REM  computer's move
110  PRINT "COMPUTER TAKES ";C
120 M = M - C: REM  remaining matches
130  PRINT M;" MATCHES REMAIN."
140  IF M = 0 THEN  PRINT "YOU LOSE.": END
150  PRINT "HOW MANY MATCHES DO YOU WANT? ";
160  GET P
170  IF P < 1 OR P > 2 OR P > M THEN 160
180  PRINT P
190 M = M - P: REM  deduct player's choice
200  IF M > 0 THEN 90: REM  any matches left?
210  PRINT "YOU WIN!"
220  END
```

```
]RUN
 HERE I HAVE TWENTY MATCHES.
EACH PLAYER MAY TAKE EITHER
1 OR 2 AT A TIME.  THE PLAYER
WHO TAKES THE LAST MATCH WINS.
YOU GO FIRST.
20 MATCHES REMAIN.
HOW MANY MATCHES DO YOU WANT? 2
COMPUTER TAKES 2
16 MATCHES REMAIN.
HOW MANY MATCHES DO YOU WANT? 2
COMPUTER TAKES 1
13 MATCHES REMAIN.
HOW MANY MATCHES DO YOU WANT? 2
COMPUTER TAKES 1
10 MATCHES REMAIN.
HOW MANY MATCHES DO YOU WANT? 2
COMPUTER TAKES 1
7 MATCHES REMAIN.
HOW MANY MATCHES DO YOU WANT? 2
COMPUTER TAKES 1
4 MATCHES REMAIN.
HOW MANY MATCHES DO YOU WANT? 2
COMPUTER TAKES 1
1 MATCHES REMAIN.
HOW MANY MATCHES DO YOU WANT? 1
YOU WIN!

]
```

The random number generated at line 60 determines who will go first. If the computer goes first, then lines 70 and 80 are executed, forcing the computer to select 2 matches. Otherwise, if the player goes first, control is passed to line 130. A GET is used at line 160 to input the player's choice. Note that a PRINT statement is used at line 180 so that the player can see his or her response. Remember that the GET statement does not print the keyboard input. Line 170 insures that only 1 or 2 matches are taken by the player and that the number of matches selected does not exceed the number of matches remaining. The computer's choice, based on the player's previous choice, is calculated at line 100. The sum of the two moves should be 3.

Program 3.11

This program simulates a roulette wheel, which has two colors (red and black) and thirty numbers. The player is asked to make a bet and pick a color and number. Picking the correct color returns the bet; picking the correct number but wrong color wins thirty times the bet, and picking the correct number and color wins the player sixty times the bet. Note that the maximum bet allowed is $1000. Why would a casino using this roulette wheel be highly unprofitable?

```
1   REM  N = number chosen by player
2   REM  C = conversion of C$ to a number (1 or 2)
3   REM  B = player's bet
4   REM  M = money made by player
5   REM  N1 = roulette wheel number
6   REM  C1 = roulette wheel color
7   REM  C$ = color chosen by player
8   REM
10  REM  prompt player for guess & bet
20  INPUT "WHAT NUMBER (1-30)? ";N
30  PRINT "WHAT COLOR (R)ED OR (B)LACK? ";
40  GET C$
50  PRINT C$:C = 2: REM  assume black
60  IF C$ = "R" THEN C = 1: REM  red chosen
70  INPUT "WHAT IS YOUR BET? ";B
80  IF B < 0 THEN  PRINT "NO CHEATING!": GOTO 70
90  IF B > 1000 THEN  PRINT "HOUSE LIMIT: $1000": GOTO 70
100  REM
110  REM  spin roulette wheel
120 N1 =  INT (30 *  RND (1) + 1)
130 C1 =  INT (2 *  RND (1) + 1)
140  REM  simulate the spin on the screen
150  HOME
160  FOR I = 1 TO N1
165  VTAB 12: HTAB 15
170  PRINT I;"   BLACK";
175  FOR D = 1 TO 100: NEXT D
180  HTAB 19: PRINT " RED   ";
185  FOR D = 1 TO 100: NEXT D
190  NEXT I
200  IF C1 = 2 THEN  HTAB 20: PRINT "BLACK"
210  PRINT
220  REM
300  REM  compute wins or losses
310 M =  - B: REM  assume player lost
320  IF N1 = N AND C1 = C THEN 360
330  IF N1 = N THEN  PRINT "YOU GOT THE NUMBER":M = 30 * B: GOTO 410
340  IF C1 = C THEN  PRINT "YOU GOT THE COLOR":M = B: GOTO 410
350  PRINT "YOU LOSE. BETTER LUCK NEXT TIME!": GOTO 410
360  PRINT "CONGRATULATIONS, YOU GOT BOTH!":M = 60 * B
370  REM
400  REM  tell player of wins or losses
410  IF M < 0 THEN  PRINT "YOU LOST $"; - M: GOTO 430
420  PRINT "YOU NOW HAVE $";M + B
430  PRINT "WHY NOT TRY AGAIN?": END
```

```
]RUN
WHAT NUMBER (1-30)? 25
WHAT COLOR (R)ED OR (B)LACK? R
WHAT IS YOUR BET? 100

                 15 RED

YOU GOT THE COLOR
YOU NOW HAVE $200
WHY NOT TRY AGAIN?

]RUN
WHAT NUMBER (1-30)? 14
WHAT COLOR (R)ED OR (B)LACK? R
WHAT IS YOUR BET? 200

                 7   BLACK

YOU LOSE. BETTER LUCK NEXT TIME!
YOU LOST $200
WHY NOT TRY AGAIN?

]
```

Lines 20 through 70 prompt the gambler for a number between 1 and 30, a color (red or black), and his or her bet. Line 80 insures that the gambler does not attempt to defraud the bank by placing a negative bet. This simulated roulette table has a $1000 limit, which is strictly enforced by line 90. The actual 'spin' of the roulette wheel takes place at lines 120 and 130. The player's winnings (or losses) are computed in lines 310 through 360. Note that line 360 is reached only if both conditions are satisfied in the IF . . . THEN statement at line 320, which determines if both the color and number were matched by the player. Line 350 is reached only if the player incorrectly guessed both the color and the number, conditions which are checked by lines 330 and 340. Lines 410 and 420 inform the gambler of his or her good (or bad) fortune.

Note that a GET is used at line 40 rather than INPUT so that only a single letter can be entered. If INPUT were used and the user responded by typing 'RED' rather than just 'R' as instructed, the comparison at line 60 would fail.

Exercises

1. Generate ten random numbers between 0 and 1, but print only those which are greater than 0.5.

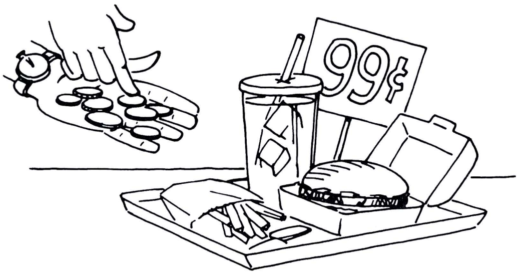
2. Generate three random numbers between 0 and 1, and print their sum.

3. Input a number (N). Print it only if it is an integer. (Hint: compare N with INT(N)).

4. Allow a user to guess a random integer between -3 and 4, inclusive. Print whether the guess was correct or not. If the guess was wrong, the correct value is also to be printed.

5. Determine randomly how many coins you find on the street. You are to find from 2 to 5 nickels, 1 to 4 dimes and 0 to 3 quarters. Lunch costs 99 cents. The program is to report the amount that you found and whether you are able to buy lunch with it.

```
]RUN
You found $.45
Sorry, you can't buy lunch

]RUN
You found $1.25
You can buy lunch

]
```



6. Input an integer (N), and print the sum of N random numbers between 0 and 1. Also print N/2 for comparison with the sum.

7. A child puts pennies into a piggy bank once each week for four weeks. The bank already contains 11 pennies in it when the child first receives it. Write a program to allow pennies to be added each week and to print the dollar value of the bank's contents after each addition.

8. Write a program that will print ten random integers from −10 to 10, inclusive.

9. Use HTAB and VTAB to draw a straight diagonal line (composed of asterisks) starting at the upper left hand corner and moving one space over for each space down until the line hits the bottom of the screen.

10. Write a program to flip a coin 50 times and print the total number of heads and tails. Run the program ten times to get a comparison among the runs.

11. Brad Browne wrote both of the following programs but has forgotten what their output is. Determine the output and check your answer by running the program.

(a)

```
10  FOR Z = 1 TO 4
20  PRINT  TAB( Z);Z
30  NEXT Z
40  FOR Y = 1 TO 4
50  PRINT  TAB( Y),Y
60  NEXT Y
```

(b)

```
10 G = 123.456
20  PRINT  INT (10 * G) / 10, INT (10 * G + .5) / 10
30  PRINT  INT (100 * G) / 100, INT (100 * G + .5) / 100
```

12. Write a program that generates 10 random integers betwen 8 and 25, inclusive, and prints them on the same line. The output should be similar to the following:

```
17  12  22  25  8  17  19  11  21  23
```

13. Make a chart showing in their correct order the values taken by the variables X and Y. Circle those values that are printed by the computer. Check by running the program.

```
10  FOR X = 1 TO 3
15  READ Y
20  IF Y > 0 THEN 35
25 Y = Y + X
30  IF X = 2 THEN Y = Y - 1: GOTO 40
35  PRINT X,Y
40  NEXT X
45  DATA  5,0,-1
```

14. (a) Determine a possible output for the program below:
(b) Rewrite the program so that the message LEARN THE MULTIPLICATION TABLE is printed if three wrong answers are given:
(c) Rewrite the program so that five different questions are asked and the message NICE GOING, HOTSHOT is printed if all five questions are answered correctly:

```
10  HOME : VTAB (10)
20 A =  INT (10 *  RND (1) + 1)
30 B =  INT (10 *  RND (1) + 1)
40  PRINT  TAB( 12);A;" * ";B;" = ";
50  INPUT C
60  IF C = A * B THEN 90
70  PRINT "You are wrong. Try again."
80  GOTO 40
90  PRINT  TAB( 12)
100  FLASH : PRINT "CORRECT": NORMAL
```

15. Write a program that contains one FOR . . . NEXT loop which finds the sum of all the odd integers from 13 to 147, inclusive. The output should be as follows:

```
THE SUM = 5440
```

16. Suzy Sunshine wrote the following program. It allows the user to enter at the keyboard any integer greater than 1 and to have the computer tell the user whether or not the integer is prime. A prime number is an integer that contains only itself and 1 as factors. The computer tests the integer by repeatedly dividing it by integers smaller than itself but larger than 1 and checking whether the quotient is whole. If so, the integer entered by the user is not prime. The program contains three errors. Find them, rewrite the program and run it. The output should look like this:

```
]RUN
Integer > 2 please? 12
That integer is not prime.

]RUN
Integer > 2 please? 17
That integer is prime.

]
```

```
20  READ "Integer > 2 please ";N
30  FOR X = 2 TO N - 1
40  IF N / X =  INT (N / X) THEN 70
50  NEXT N
60  PRINT "That integer is prime."
70  PRINT "That integer is not prime."
```

17. Generate 1000 random integers between 1 and 9, inclusive, and print how many were even and how many were odd. The output should be similar to the following:

```
]RUN
There were 577 odd integers.
There were 423 even integers.

]
```

18. Write a program that allows the user and computer to select alternately integers between 3 and 12, inclusive, keep a sum of all the integers selected and declare the winner to be the one who selects that integer which makes the sum greater than 100. Have the program ask the user if he or she would like to proceed first or second.

19. A bank pays interest once a year at a yearly rate of 5%. A man deposits $1000 on January 1, 1984 and wishes to leave it there to accrue interest until the balance is at leat $2000. Compute the balance on Jan. 1 of each year, starting with 1985 and ending in the year when the balance exceeds $2000. The output should resemble the following:

```
Date              Balance
Jan 1, 1985       $1050
Jan 1, 1986       $1102.5
...               ...
...               ...
...               $1979.93
...               $2078.93
```

20. Input a positive integer N, and print all positive integers that are factors of N. The output should resemble the following:

```
]RUN
A positive integer, please? 1.4
Your number is not a positive integer.
A positive integer, please? 12
1 2 3 4 6 12
]
```

21. Print the radius (cm.) of a sphere, given its volume (cm.^{3}). Round off the results to the nearest hundredth:

DATA: 690, 720, 460, 620
Note: Volume $= (4/3)\ (\pi)\ R^3$, where $\pi = 3.14159$

22. (a) Print twenty random integers between 0 and 100, inclusive.
(b) Change the program so that sixty percent of the twenty integers printed will be less than 25.

23. Using the POS function, write a program that determines the length of a string.

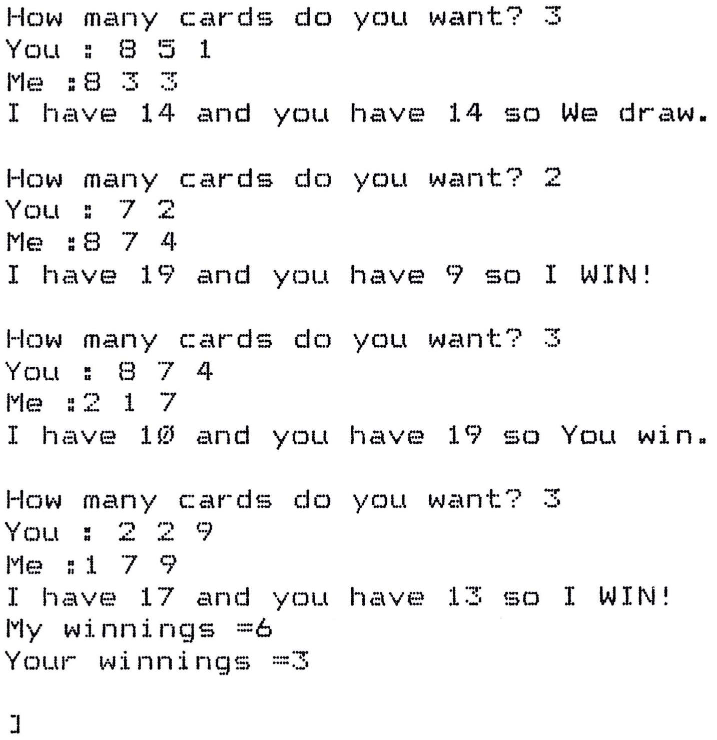
```
]RUN
Your string? OZ-1335
OZ-1335
The length of OZ-1335 is 7

]
```

24. Put a flashing notive on the screen which advertises Uncle Bill's whamburgers for $0.59.

```
]RUN
-----------------------
!                     !
! TODAY ONLY!         !
!     UNCLE BILL'S    !
!     WHAMBURGERS     !
!          ONLY $0.59 !
!                     !
-----------------------

]
```

25. Write a program that will pick 20 random integers from 1 to 100, inclusive, and print only those numbers that are prime. A prime number is one that can be only divided exactly (i.e. no remainder) by itself and 1.

26. You are to write a program to simulate a simplified version of the game 21. A deck of cards numbered from 1 to 10 are used and any number can be repeated since the deck contains many cards. the computer starts by asking you how many cards you want. It then deals you the cards, which are randomly picked, and totals their value. If the value exceeds 21, you automatically lose. If your value is under 21, the computer deals itself *three* randomly picked cards. The winner is the one with the highest score equal to or less than 21. Write your program so that the game can be played 10 consecutive times with the winner of each game winning one dollar. At the end of the 10 games print out the total winnings for you and the computer. The example below shows only the last four of the ten games:



```
How many cards do you want? 3
You : 8 5 1
Me :8 3 3
I have 14 and you have 14 so We draw.

How many cards do you want? 2
You : 7 2
Me :8 7 4
I have 19 and you have 9 so I WIN!

How many cards do you want? 3
You : 8 7 4
Me :2 1 7
I have 10 and you have 19 so You win.

How many cards do you want? 3
You : 2 2 9
Me :1 7 9
I have 17 and you have 13 so I WIN!
My winnings =6
Your winnings =3

]
```

27. Have the computer randomly select a number of quarters (from 0 to 7), dimes (from 0 to 4) and pennies (from 0 to 9) and print the exact number of coins and their total value. The user has three chances to determine how many quarters, dimes and pennies were selected.

28. (a) Have the Apple print the dart board pictured below, and have it fire ten random shots, using an asterisk to indicate the spot hit. The top of the outer square is on line 3 and its two vertical sides are at TAB(10) and TAB(30).

```
]RUN
         ---------------------
         !                   !
         !                   !
         !                   !
         !                   !
         !       -----       !
         !       !   !       !
         !       !   !       !
         !       !   !       !
         !       !   !       !
         !       -----       !
         !                   !
         !                   !
         !                   !
         !                   !
         ---------------------

]
```

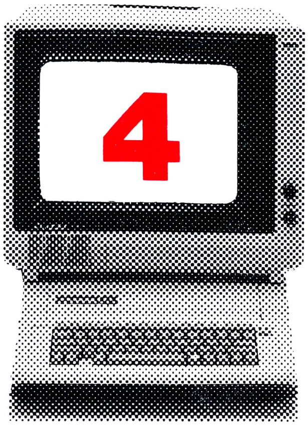
(b) Rewrite program (a) to give the score at the end of a game. A hit in the center square is worth 10 points, the outer square 4 points, and the region outside both −1 points.
(c) Rewrite program (b) so that all hits inside either square flash.

29. Using HTAB and VTAB, write a program that will draw a football made of asterisks. (Hint: use the equation of a circle.)

```
     *
  *     *
 *       *
*         *
*         *
*         *
*         *
*         *
 *       *
  *     *
     *
```

30. Modify the answer to Exercise 29 so that the left half of the figure is flashing.

31. Luke Skywalker needs a simple simulator for his APPLE so that he can train before his next battle against the Empire. Help Luke by writing a simple Star Wars game. Have the computer print two asterisks at random positions on the screen. Luke has ten chances to guess the X and Y coordinates of the two enemy asterisks or the Rebellion will fail. The computer should flash a message at the end of the game stating the outcome. Use HTAB and VTAB to position to input line every time so that the enemy asterisks are not pushed off the top of the screen during the course of the game.



NESTED LOOPS AND SUBSCRIPTED VARIABLES

DIM

CLEAR

DIM

CLEAR

DIM

CLEAR

DIM

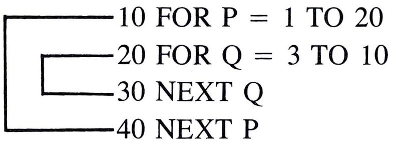
CLEAR

We have previously observed that the FOR . . . TO . . . STEP, NEXT statements set up loops. This chapter proceeds further by showing how to combine two or more loops in such a way as to place one loop inside another.

The second part of this chapter deals with subscripted variables, which use a fixed variable name in conjunction with a variable subscript (e.g., A(N), where A is fixed and N varies). This technique enables a program to deal conveniently with a large amount of data. Subscripted variables usually employ FOR . . . NEXT loops to generate the values for the subscripts.

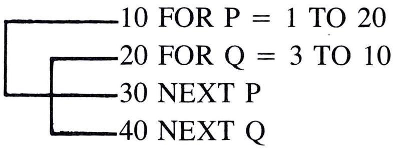
Nested FOR . . . NEXT Loops

The concept of a FOR . . . NEXT loop was presented in Chapter Two. In this chapter, the concept of nested FOR . . . NEXT loops (that is, one loop placed, or 'nested' within another) is introduced. For example:



```
10 FOR P = 1 TO 20
20 FOR Q = 3 TO 10
30 NEXT Q
40 NEXT P
```

By definition nested loops must never cross. One loop must be contained entirely within another or be entirely separated from another. For example, the following arrangement is not permissible because the loops cross each other:



```
10 FOR P = 1 TO 20
20 FOR Q = 3 TO 10
30 NEXT P
40 NEXT Q
```

When this program is run, the following error message will result:

```
?NEXT WITHOUT FOR ERROR IN 40
```

Program 4.1

This program uses nested loops to print a portion of the multiplication table:

```
10  FOR X = 1 TO 5
20  FOR Y = 2 TO 4
30  PRINT X;"*";Y;"=";X * Y,
40  NEXT Y
50  PRINT
60  NEXT X

]RUN
1*2=2               1*3=3               1*4=4
2*2=4               2*3=6               2*4=8
3*2=6               3*3=9               3*4=12
4*2=8               4*3=12              4*4=16
5*2=10              5*3=15              5*4=20

]
```

Line 10 establishes the outer X loop and starts X with the value 1. X retains this value until it is incremented by the execution of line 60. Line 60 is not executed, however, until the inner Y loop, lines 20-40, has run its entire course. X therefore retains the value 1 while Y changes from 2 to 4. When the Y loop has finished its cycle of two passes, line 60 is executed, incrementing X to 2. The program returns to line 20 and starts the Y loop over again at its initial value of 2. Whenever a program encounters the FOR . . . TO statement of a loop, the loop variable is reset to its initial value. Since this program does not return to line 10, the X loop is never reset to 1. Notice that Y will take on its values of 2 and 4 five times.

Program 4.2

This program calculates and prints all possible combinations of quarters, dimes and nickels that add up to fifty cents:

```
10  PRINT "QUARTERS","DIMES","NICKELS"
20  FOR Q = 0 TO 2: REM  QUARTERS
30  FOR D = 0 TO 5: REM  DIMES
40  FOR N = 0 TO 10: REM  NICKELS
50  IF Q * 25 + D * 10 + N * 5 = 50 THEN  PRINT Q,D,N
60  NEXT N
70  NEXT D
80  NEXT Q

]RUN
QUARTERS        DIMES           NICKELS
0               0               10
0               1               8
0               2               6
0               3               4
0               4               2
0               5               0
1               0               5
1               1               3
1               2               1
2               0               0

]
```

Q represents the number of quarters, D the number of dimes and N the number of nickels. There may be anywhere from 0 to 2 quarters in a combination. Similarly, there may be anywhere from 0 to 5 dimes and from 0 to 10 nickels. Three nested loops ensure that every possible combination that *might* work is checked at line 50.

Review

1. Use nested loops to produce the following output. The outer loop runs from 20 to 24, and the inner loop runs from 1 to 3:

```
]RUN
OUTER: 20
INNER:1              INNER:2              INNER:3
OUTER: 21
INNER:1              INNER:2              INNER:3
OUTER: 22
INNER:1              INNER:2              INNER:3
OUTER: 23
INNER:1              INNER:2              INNER:3
OUTER: 24
INNER:1              INNER:2              INNER:3

]
```

The Need for Subscripted Variables

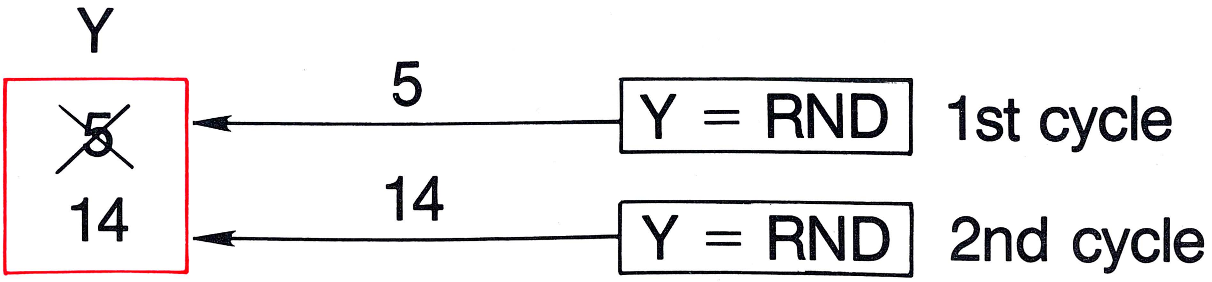
The major purpose of subscripted variables is to store easily and later retrieve large quantities of data. Programs 4.3 and 4.4 do not use subscripted variables but demonstrate when contrasted with later programs how useful subscripted variables are.

Program 4.3

This program generates ten random numbers from one to twenty, inclusive:

```
10  FOR X = 1 TO 10
20 Y =  INT (20 *  RND (1) + 1)
30  PRINT Y;"   ";
40  NEXT X


]RUN
5   14   1   16   20   13   20   11   17   17
]
```

Every time a new value is assigned to Y, it replaces the previous value for Y. Since the previous values of Y are not remembered, it is impossible to prevent repetition by comparing the old values of Y with the new value of Y. If a box analogy is used here, the first two cycles of Program 4.3 would appear as follows:



Program 4.4

This program prints four random numbers between 1 and 20, inclusive, without any repetition. The technique can be expanded to have the program choose 10 numbers if the user is willing to type the program lines required:

```
10 Y1 =  INT (20 *  RND (1) + 1)
20  PRINT Y1;" ";
30 Y2 =  INT (20 *  RND (1) + 1)
40  IF Y2 = Y1 THEN 30
50  PRINT Y2;" ";
60 Y3 =  INT (20 *  RND (1) + 1)
70  IF Y3 = Y2 OR Y3 = Y1 THEN 60
80  PRINT Y3;" ";
90 Y4 =  INT (20 *  RND (1) + 1)
100  IF Y4 = Y3 OR Y4 = Y2 OR Y4 = Y1 THEN 90
110  PRINT Y4

]RUN
18 1 17 10

]
```

The similarity between the three sets of lines 30-50, 60-80 and 90-110 is obvious. In each set the first line selects a random number between 1 and 20, inclusive, and the second line checks to see if the number is a repetition of a number previously chosen. If this is the case, then execution goes back to the first line in the set so that another number may be chosen. Finally, the third line of the set prints the random number selected. In Program 4.6, the use of subscripted variables will eliminate the need for the repetition of these sets.

Single Subscripted Variables

In mathematics a set of single subscripted numeric variables (e.g., X_1, X_2, X_3, . . .) is symbolized by a letter and a subscript, which is written below the line of the letter. On the computer the same set of subscripted variables would be referred to as X(1), X(2), X(3) and so on, where the integer enclosed in parentheses is the subscript.

The name for a set of single subscripted numeric variables consists of one of the usual numeric variable names, followed by the parenthesized subscript. For example, A(1), B2(17) and X(8) are all legal subscripted variable names. Similarly, single subscripted string variables names such as Z$(5), Y1$(20) and M$(14) are acceptable.

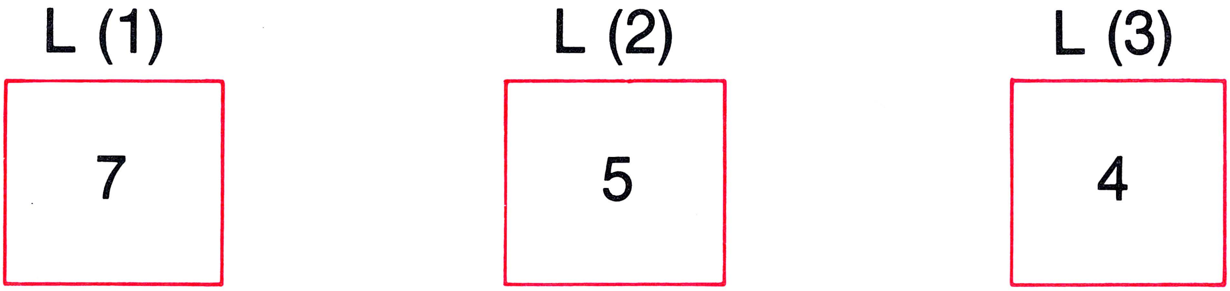
The subscript variable L(2) is *not* the same as the variable L2. Furthermore, the subscript is a part of the variable name and must not be confused with the value of the variable. For example, in the statement L(5) = 32 the subscript is 5 and the value of L(5) is 32.

Program 4.5

This program illustrates the difference between the subscript and the value of a subscripted variable:

```
10 L(1) = 7
20 L(2) = 5
30 L(3) = 4
40  PRINT "L(1)=";L(1),"L(2)=";L(2),"L(3)=";L(3)
50  PRINT "L(1 + 2) = ";L(1 + 2)
60  PRINT "L(1) + L(2) = ";L(1) + L(2)
70 X = 2
80  PRINT "L(X) = ";L(X)

]RUN
L(1)=7           L(2)=5           L(3)=4
L(1 + 2) = 4
L(1) + L(2) = 12
L(X) = 5

]
```

Lines 10 through 30 set the value of L(1) through L(3) as follows:



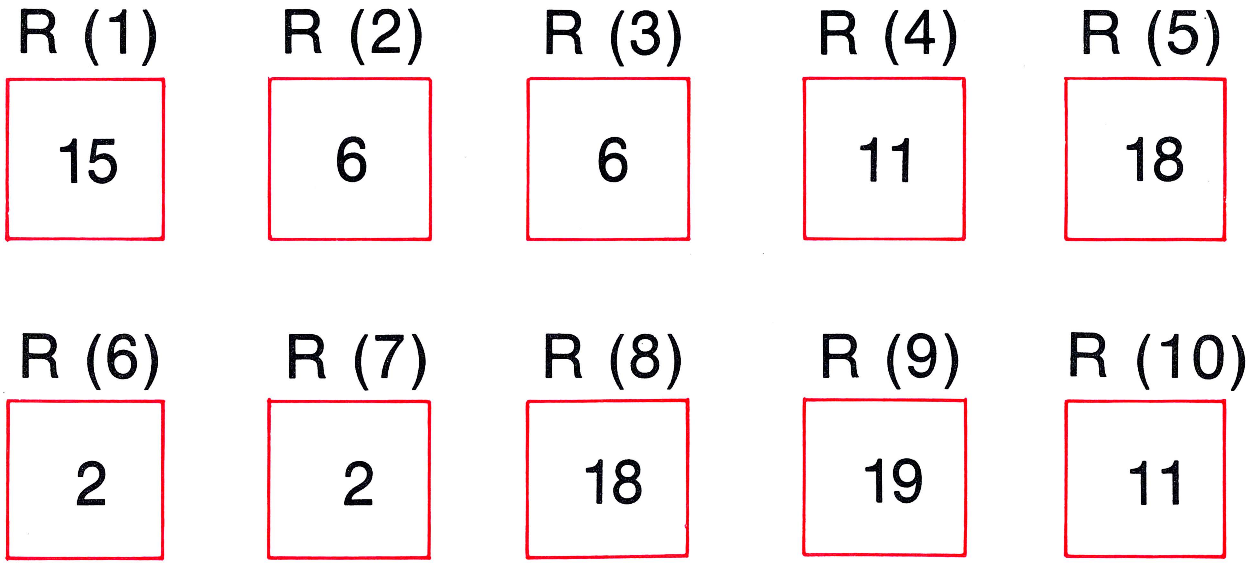
Lines 50 and 60 point out the difference between adding two subscripts and adding the values of two variables. L(1 + 2) is identical to L(3) and therefore has a value of 4. L(1) + L(2) calls for the values 7 and 5 to be added, thus producing a total of 12. Since the subscript X equals 2 in line 80, a value of 5 is printed. As can be seen, it is permissible to use a numeric variable as the subscript of a subscripted variable.

Program 4.6

Like Program 4.3, this program selects 10 random numbers between 1 and 20 and makes no attempt to prevent repetition. Yet unlike Program 4.3, it stores the numbers chosen in a subscripted variable:

```
10   FOR X = 1 TO 10
20  R(X) =  INT (20 *  RND (1) + 1)
30   NEXT X
40   PRINT "TEN RANDOM NUMBERS HAVE BEEN STORED IN R()."
50   PRINT
60   FOR N = 1 TO 10
70   PRINT "R(";N;") HAS A ";R(N);" STORED IN IT."
80   NEXT N

]RUN
TEN RANDOM NUMBERS HAVE BEEN STORED IN R().

R(1) HAS A 15 STORED IN IT.
R(2) HAS A 6 STORED IN IT.
R(3) HAS A 6 STORED IN IT.
R(4) HAS A 11 STORED IN IT.
R(5) HAS A 18 STORED IN IT.
R(6) HAS A 2 STORED IN IT.
R(7) HAS A 2 STORED IN IT.
R(8) HAS A 18 STORED IN IT.
R(9) HAS A 19 STORED IN IT.
R(10) HAS A 11 STORED IN IT.

]
```

This program has two loops which are not nested but which follow one another. Each loop is executed 10 times. In the first loop line 20 chooses a random number and stores it in one of the subscripted R() variables. The first time through this loop (X = 1) a random number is stored in R(1); the second time (X = 2), a new number is stored in R(2) and so on repeating 10 times. After the above run of this program, the R() boxes had contents as follows:



The second loop (lines 60 through 80) prints the contents stored in the boxes.

The ability to store numbers in this way will allow the writing of a new program to choose 10 random numbers *without* repetition. By checking whether a chosen random number equals any of those previously selected, a program can determine whether to accept the random number chosen or to make another selection.

Program 4.7

This program uses nested loops and subscripted variables to pick 10 random numbers between 1 and 20 without repetition:

```
10  FOR X = 1 TO 10
20 Y(X) =  INT (20 *  RND (1) + 1)
30  IF X = 1 THEN 70
40  FOR Q = 1 TO X - 1
50  IF Y(X) = Y(Q) THEN 20
60  NEXT Q
70  PRINT Y(X);"  ";
80  NEXT X

]RUN
20  13  17  15  11  4  14  3  6  1
]
```

Line 20 selects a random number between 1 and 20, inclusive, and stores it in one of the subscripts of Y(). Since Y(1) is the first random number, line 30 is included to insure Y(1) will be printed immediately because it is obviously not a repeat of another number. Lines 40 to 60 constitute a nested loop which determines if a number just chosen, Y(X), is equal to any of the previously chosen numbers, Y(1) through Y(X-1). The X-1 in line 40 insures that Y(X) is not rejected by being checked against itself. If X rather than X-1 were used, then Q would eventually equal X. Line 50 would determine that Y(X) was a repeated number when Q equalled X and would then mistakenly return to line 20.

There is an additional reason for including line 30. When X and Q are equal to 1, Y(X) will equal Y(Q), and without line 30, program flow would continuously return to line 20.

Program 4.8

The following program will randomly read a list of 5 names into a subscripted string variable N$() without repeating any of the names:

```
10  FOR X = 1 TO 5
20 Y =  INT (5 *  RND (1) + 1)
30  IF N$(Y) <  > "" THEN 20
40  READ N$(Y)
50  NEXT X
60  FOR Z = 1 TO 5
70  PRINT N$(Z)
80  NEXT Z
90  DATA  TED,JOHN,MARY,DON,SUSAN

]RUN
JOHN
DON
TED
MARY
SUSAN
```

```
]RUN
MARY
TED
JOHN
DON
SUSAN

]
```

This program reads the names in the DATA statement in order but randomly places them in N$(1) to N$(5). Repetition is avoided by checking each new box as it is selected to discover whether it contains a name. If it is full, a new box is tried by selecting a new random number. The consecutive double quotation marks ("") at line 30 are used to check whether the box is empty. Unassigned subscripted string variables, which are empty, are represented on the computer by ("").

Review

2. Using subscripted variables, write a program in which 3 numbers are input. Then have the computer type them back in reverse order.

```
]RUN
?4
?6
?1
1
6
4

]
```

3. Six words are to be entered from the keyboard. Have the computer randomly select and print four of the words as a "sentence" (which may or may not make sense). Repetition of words is allowed.

```
]RUN
?JACK
?AND
?JILL
?RAN
?AWAY
?SCREAMING
SCREAMING JILL AND AND .

]
```

4. Modify the program of the previous exercise so that words are not repeated.

```
]RUN
?JACK
?AND
?JILL
?RAN
?AWAY
?HAPPILY
JACK JILL AND AWAY.

]
```

DIM

Whenever more than 10 elements are needed in a subscripted variable, the computer must be informed. The DIM statement is used to direct the computer to reserve enough space in memory to accommodate the anticipated input. Its format is:

```
DIM <variable>(<dimension>)
```

For example, Program 4.8 could be changed to store 100 names by making the appropriate changes in several lines and adding the following line:

```
5 DIM N$(100)
```

The dimension may be either a numeric constant or a numeric variable. For example,

```
5 N = 100 : DIM N$(N)
```

is equivalent to line 5 above.

If the ages of the people were also to be stored, a numeric subscripted variable would have to be added and the DIM statement modified:

```
5 DIM N$(100), A(100)
```

It is possible to request more space (i.e. boxes) in memory than the computer can supply. This results in the following error message:

```
?OUT OF MEMORY ERROR
```

It is a good idea to place DIM statements at the beginning of a program because the DIM statement must be executed before more than 10 subscripts of a subscripted variable have been used. Though it is possible to have several DIM statements in a program, each subscripted variable may only be dimensioned once. If there are two DIM statements for the same variable or if a DIM statement is executed more than once, the error message

```
?REDIM'D ARRAY ERROR
```

will result.

CLEAR

In programs where available memory usage is limited it is sometimes useful to be able to clear variables which are no longer needed. This can be done using the CLEAR statement. For example,

```
100 CLEAR
```

will remove all variables, including subscripted variables, from memory.

Program 4.9

This program demonstrates the effect of CLEAR on variables stored in the computer's memory:

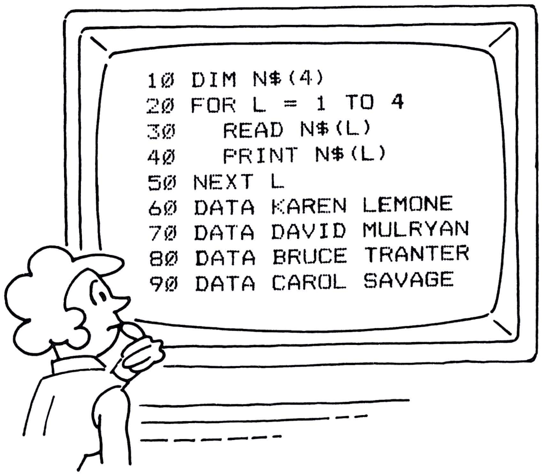
```
10 A = 47.9
20 R$ = " THIS IS AN EXAMPLE OF CLEAR "
30 T(4) = 3.14159
40  PRINT A;R$;T(4)
50  CLEAR
60  PRINT
70  PRINT A;R$;T(4);" WHAT HAPPENED TO THE VARIABLES?"

]RUN
47.9 THIS IS AN EXAMPLE OF CLEAR 3.14159

00 WHAT HAPPENED TO THE VARIABLES?

]
```

Double Subscripted Variables

The computer can also use double subscripts to name a variable. This is similar to single subscripting except that there are two numbers within the parentheses instead of one. For example,

A(1,5)   B3(7,3)   C$(4,9)

are all double subscripted variables. The programmer can visualize the space in memory for a double subscripted variable as an array of rows and columns, rather than a single row (as in the case with a single subscripted variable). This procedure provides a convenient technique for dealing with problems in which the data are two-dimensional in nature, such as the location of seats in a theater.



```
10 DIM N$(4)
20 FOR L = 1 TO 4
30   READ N$(L)
40   PRINT N$(L)
50 NEXT L
60 DATA KAREN LEMONE
70 DATA DAVID MULRYAN
80 DATA BRUCE TRANTER
90 DATA CAROL SAVAGE
```

The box analogy will help the programmer understand more clearly how the double subscripted variable operates. The first integer in the subscript identifies the row and the second the column in which the variable is located. For example, A(2,3) is located at the second row, third column:

	Col. 1	Col. 2	Col. 3	Col. 4
Row 1	A (1, 1)	A (1, 2)	A (1, 3)	A (1, 4)
Row 2	A (2, 1)	A (2, 2)	A (2, 3)	A (2, 4)
Row 3	A (3, 1)	A (3, 2)	A (3, 3)	A (3, 4)

Program 4.10

A classroom has 5 rows of seats with 3 seats to a row. The following program randomly assigns a class of 14 students to seats, leaving one seat empty:

```
10  DIM N$(5,3)
20  REM  read 14 names randomly into the array N$
30  FOR X = 1 TO 14
40  READ A$
50 R =  INT (5 *  RND (1) + 1)
60 C =  INT (3 *  RND (1) + 1)
70  IF N$(R,C) <  > "" THEN 50
80 N$(R,C) = A$
90  NEXT X
100  REM  print the seating chart
110  FOR R1 = 1 TO 5
120  FOR C1 = 1 TO 3
130  IF N$(R1,C1) = "" THEN  PRINT "EMPTY",: GOTO 150
140  PRINT N$(R1,C1),
150  NEXT C1
160  NEXT R1
170  REM
200  DATA  ANNE, DON, SHERRY, MAGGIE, TED, LIZ, ROB
210  DATA  MARY, DAVID, MARK, KEVIN, SUSAN, WENDELL, CINDY
```

```
]RUN
MARY                SHERRY              EMPTY
CINDY               DON                 ANNE
DAVID               LIZ                 SUSAN
WENDELL             ROB                 KEVIN
MARK                TED                 MAGGIE

]
```

The computer also allows 3, 4, etc., all the way up to 46 dimensions in a subscripted variable (e.g., DIM X(5,5,8,2,3,1)). However, the more dimensions that a subscripted variable has, the smaller each dimension must be because of the limited amount of space available in the computer's memory.

Review

5. Six numbers are to be input from the keyboard and stored in the subscripted variable X(). These are subsequently to be printed in a vertical column, and then a second time, closely spaced on a single line.

```
]RUN
?23
?67
?128
?37
?42
?143
23
67
128
37
42
143

23 67 128 37 42 143
]
```

6. Use the double subscripted variable X$(I,J) for which the row variable I runs from 1 to 5 and the column variable J from 1 to 3. Enter the letters A, B, C as the first row, D, E, F as the second, up to M, N, O as the fifth. Have the program print the following, making the rows become columns (Hint: Use READ, DATA):

```
]RUN
A D G J M
B E H K N
C F I L O

]
```

Some Final Notes on Subscripted Variables

Subscripted string and numeric variables greatly enhance the programmer's ability to store and deal with large quantities of data within any one program run. It must be remembered, however, that if the program is run again, all of the stored data in the computer's memory is erased. This means that all of the boxes become either null ("") or zero at the start of the next run. A method for permanently storing data is presented in Chapter Nine.

Extended Variable Names

Previously, variable names have consisted of a single letter or a single letter followed by a digit. As was pointed out in this chapter, these names may also be subscripted. Actually, variable names may be of any length. For example, ALPHABET, HAROLD, SCARE and BLACKBEARD23 are all permissible variable names. There are, however, drawbacks in these names. Nowhere in the name of a variable may any key word occur. For example, SCORE, FORTUNE, SAND and DIMWIT are all unacceptable variable names because they contain the words OR, FOR, AND and DIM, respectively. Also, no matter how long the variable name may be, the Apple considers only the first two characters. This means the variable names XMIN and XMAX are considered the same because they both start with XM. Because of these drawbacks, it is recommended that extended variable names not be used except where they are useful in identifying what a variable represents.

Exercises

1. Using a nested loop, have the computer print a rectangle consisting of eight lines of thirty asterisks each.

2. Show the output of the following program and check by running it. Rerun the program after removing line 50:

```
10  FOR I = 1 TO 5
20  FOR J = 1 TO 2 * I - 1
30  PRINT ".";
40  NEXT J
50  PRINT
60  NEXT I
```

3. Enter values of X(I) for I = 1 to 6. Print the values of I and X(I) in two columns with I proceeding in the order 1, 3, 5, 2, 4, 6.

4. Enter 15 letters of the alphabet (not necessarily different) and print them in reverse order as a single block of letters.

5. Have the computer compute the values of A(I,J), where A(I,J) = 3 * I + J * J, I varies from 1 to 4, and J varies from 1 to 12. The user is to input a number N from 1 to 4 so that all values of A(N,J) can be printed.

6. Using a DATA statement, have the computer enter one letter of the alphabet for each member of A$(I,J), where I runs from 1 to 11 and J from 1 to 3. The letters are first to be printed in the form of an 11 word sentence, each word consisting of 3 letters. Then, the letters are to be printed again as a 3 word sentence, each word consisting of 11 letters. The words may or may not make sense.

7. Carol, who has forgotten where the computer is, needs to know the output for the following programs. Predict the output in each case and check by running the program.

(a)
```
10  FOR L1 = 1 TO 3
20  FOR L2 = 5 TO 6
30  PRINT L1,L2
40  NEXT L2
50  NEXT L1
```

(b)

```
10  FOR X = 10 TO 15 STEP 2
20  FOR Y = 15 TO 10 STEP  - 2
30  IF Y = X THEN 99
40  IF X < Y THEN  PRINT X: GOTO 60
50  PRINT Y
60  NEXT Y
70  NEXT X
99  END
```

(c)

```
10  FOR S = 1 TO 10
20  READ A(S)
30  NEXT S
40  PRINT A(3),A(7),A(10)
50  DATA  23,12,45,2,87,34,89,17,2,35,70
```

8. Suzy Sunshine has done it again and has written another program that will not run properly. Assist her by correcting the program. The output should look like this:

```
]RUN
3                4
4                3
4                4
5                3
5                4

]
```

```
10  FOR A = 3,5
20  FOR B = 1,4
30  IF A * B <  = 10 GOTO 50
40 PRINYAB
50  NEXT A
60  NEXT B
```

9. The following program is designed to print the numbers in the DATA statement in decreasing order. However, there are some errors in the program. Correct and run the program to produce this output:

```
]RUN
40 37 27 27 16 9 8 5 2 1
]
```

```
10  FOR X = 40 TO 1
20  FOR Y = 1 TO 10
30  READ N
40  IF N = X THEN  PRINT N;" ";
50  NEXT X
60  NEXT Y
70  DATA  5,27,37,16,27,8,2,40,1,9
```

10. What is the exact output for the following program?

```
10   READ B1,B2,B3,B4,B5,B6
20   FOR X = 1 TO 6
30   READ B(X)
40   NEXT X
50   PRINT "B4= ";B4;" But B(4)= ";B(4)
60   PRINT B1 + B2 + B3,
70   PRINT B(1) + B(2) + B(3),B(1 + 2 + 3)
80   DATA  3,7,4,1,8,12
90   DATA  14,42,69,86,12,111
```

11. The following program is designed to generate random integers between 1 and 99, inclusive, until it encounters a duplicate. At that point it should print how many numbers it has found and then print all of them. However, there are a number of errors in the program. Correct them to produce output similar to:

```
]RUN
Duplicate after 8 numbers
97 11 2 78 55 62 96 55
]
```

```
20   FOR X = 1 TO 100
30  N(X) =  INT (99 *  RND (1) + 1)
40   FOR Y = 1 TO 100
50   IF N(X) <  > N(Y) THEN 70
60   PRINT "Duplicate after ";X;" numbers"
70   FOR I = 1 TO X: PRINT N(X);" ";: NEXT I
80   NEXT Y
90   NEXT X
```

12. Write a program that reads the names, street addresses, towns and zip codes of five people into subscripted variables N$(X), A$(X), T$(X) and Z$(X). The user enters a name, and the program prints the full name and address of that person. If the name is not there, have the program print PERSON NOT FOUND.

13. A Pythagorean triple is a set of three integers which are the lengths of the sides of a right trangle ($C^2 = A^2 + B^2$). Find all sets of three integers up to C = 50 which are Pythagorean triples. For example, A = 3, B = 4, C = 5 is a solution. (Note that due to rounding errors, it is better to use A∗A + B∗B = C∗C rather than A↑2 + B↑2 = C↑2 to check for equalities.)

14. Stan's Grocery Store has 3 aisles and in each aisle there are five items. Write a program that will read 15 items into the subscripted variable I$(X,Y), dimensioned 3×5. Let his customers type in the item they want to buy and be informed by the computer of the aisle and the item's number.

```
]RUN
What are you looking for? Cherries
You will find Cherries in aisle #1, item #4
What are you looking for? Milk
You will find Milk in aisle #2, item #3
What are you looking for? Broccoli
I'm sorry, we don't have Broccoli.
What are you looking for?
```

15. Pick 20 random integers between 10 and 99, inclusive. Print the odd integers on one line and the even integers on the next line. The output should look like this:

```
]RUN
Odd integers: 49 25 21 13 63 77 29 77 27 43
Even integers: 66 56 18 24 28 70 56 62 48 76
]
```

16. Find the average of four grades for each of five students. The output should give in columns on consecutive lines each student's name, four grades and average. Each column should have a heading. The last student's average should be underlined and the class average printed below it in the same column.

```
]RUN
Student #1 DON
Enter four grades: 42,86,99,99
Student #2 LESTER
Enter four grades: 50,55,45,34
Student #3 LIZ
Enter four grades: 100,98,99,97
Student #4 ROB
Enter four grades: 67,72,71,68
Student #5 SUE
Enter four grades: 89,91,93,90
Name         1    2    3    4     AVE.
DON         42   86   99   99    81.5
LESTER      50   55   45   34    46
LIZ         100  98   99   97    98.5
ROB         67   72   71   68    69.5
SUE         89   91   93   90    90.75
                                 ------
                                 77.25
```

17. Marcus Welby wants you to program the computer to keep track of his busy schedule.
(a) Write a program to allow a patient to choose the day and time he or she wants to see the doctor. There are 5 days and 6 time slots for each day. If the desired slot is empty, the patient enters his or her name. If it is full, the program asks for another slot.
(b) Add the steps needed to allow Dr. Welby to print his schedule for any particular day.

18. Marilyn Monroe has 3 skirts—red, green, and purple; 2 pairs of jeans—white and electric purple; 4 blouses—blue, pink, orange, and black; and 2 sweaters—yellow and green. Have the computer print a list of all possible combinations of the articles she can wear (e.g., red skirt and blue blouse, or white jeans and yellow sweater).

RED SKIRT.....YELLOW SWEATER
GREEN SKIRT....WHITE SWEATER
PURPLE JEANS...YELLOW SWEATER
WHITE JEANS...YELLOW SWEATER
BLUE SKIRT...RED BLOUSE

19. Write an extended version of the game high-low. In this game, the computer picks a secret random number from 1 to 100 and gives the player an unlimited number of chances to guess it. For each wrong guess the computer tells whether to guess higher or lower and stores the guess in a subscripted variable. If the player guesses the same number twice, the computer should produce the message WAKE UP! You guessed that number before.

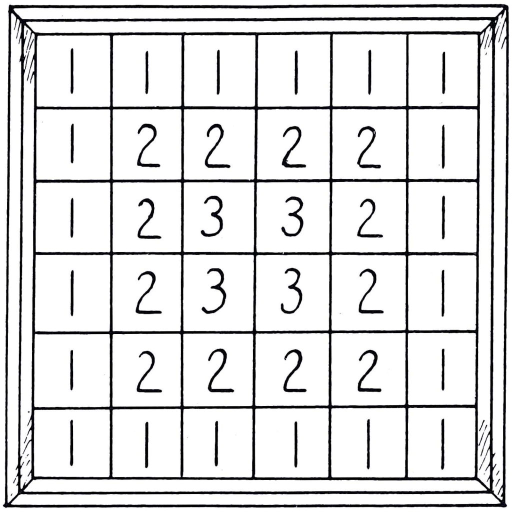
20. Betty Davis has written the following two programs to sort data. Read the programs carefully and predict the output:

(a)

```
10  DIM A(10)
20  FOR B = 1 TO 10
30  READ A(B)
40  NEXT B
50  FOR C = 2 TO 10
60  FOR D = 1 TO C - 1
70  IF A(D) <  = A(C) THEN 90
80 E = A(D):A(D) = A(C):A(C) = E
90  NEXT D
100  NEXT C
110  FOR F = 1 TO 10
120  PRINT A(F)
130  NEXT F
140  DATA  1,3,7,2,4,9,0,2,5,8
```

```
10  DIM Q$(10)
20  FOR M = 1 TO 10
30  READ Q$(M)
40  NEXT M
50  FOR R = 5 TO 10
60  FOR Z = 5 TO R - 1
70  IF Q$(Z) <  = Q$(R) THEN 90
80 T$ = Q$(Z):Q$(Z) = Q$(R):Q$(R) = T$
90  NEXT Z
100  NEXT R
110  FOR I = 10 TO 1 STEP  - 1
120  PRINT Q$(I)
130  NEXT I
200  DATA  Jon,Kim,Sue,Elizabeth,Kirstie
210  DATA  Zero,Hans,Scott,Winston,Ophelia
```

21. The game Penny Pitch is common in amusement parks. Pennies are tossed onto a checkerboard on which numbers have been printed. By adding up the numbers in the squares on which the pennies fall, a score is accumulated. Write a program which simulates such a game in which ten pennies are to be randomly pitched onto the board shown below:



1	1	1	1	1	1
1	2	2	2	2	1
1	2	3	3	2	1
1	2	3	3	2	1
1	2	2	2	2	1
1	1	1	1	1	1

Have the computer print the board with an X indicating where each penny has landed and then the score. Below is a sample run. Note that more than one penny can land in one square:

```
]RUN
1  1  1  1  X  1
1  2  2  2  2  1
X  2  3  3  2  X
1  X  3  X  2  1
1  2  2  2  X  1
X  1  1  X  1  X

Score = 14

]
```

22. Write a program that rolls two dice 1000 times and prints the number of times each different point total (2, 3, 4, 5 . . . , 12) appeared. The output should resemble the following:

```
]RUN
Point total     Times appearing
2               23
3               50
4               87
5               106
6               147
7               157
8               150
9               104
10              86
11              65
12              25

]
```

23. Write a program which makes up 15 "words" (i.e. groups of letters, whether pronounceable or not) composed of from one to seven randomly chosen letters and prints them. (Hint: use addition of strings. For example, if A$(2) = "B" and A$(12) = "L", then A$(2) + A$(12) = "BL".)

24. Use the computer to play a modified game of Othello. Have it randomly fill an 8 × 8 subscripted string variable with X's and O's and print the array by row (horizontal) and column (vertical). Examples are shown below. The X's are for player 1, the O's for player 2. Have the program ask the players alternately for the row and column of the opponent's piece that should be flipped (changed from an X to an O or vice versa). All of the opponent's piece along the horizontal or vertical line passing through the flipped piece are also flipped. For example, if player 1 flipped the O at 1,8, board A would be changed to board B.

(a)

```
    1 2 3 4 5 6 7 8
1   X X O O X O X O
2   O O O X X O X X
3   X O X X O X O O
4   O O X O X O X O
5   O O X X X O X O
6   O O X O O X X O
7   O O X X O X O X
8   X X O X X O X X
```

(b)

```
    1 2 3 4 5 6 7 8
1   X X X X X X X X
2   O O O X X O X X
3   X O X X O X O X
4   O O X O X O X X
5   O O X X X O X X
6   O O X O O X X X
7   O O X X O X O X
8   X X O X X O X X
```

25. Mr. and Mrs. Charles Windsor want to start a bank account for their newborn son, William. They open the account with $500. At the beginning of each successive year they deposit $60 more. When William is 21, how much money will be in the account? (Assume the interest rate to be 6% compounded quarterly).

PROGRAMMING TECHNIQUES

GOSUB

RETURN

ON...GOTO

ON...GOSUB

ONERR GOTO

RESUME

TRACE

NOTRACE

STOP

CONT

Writing long, complex programs may require special programming techniques. It is often helpful to divide these programs into sections, called subroutines. Subroutines are useful because they perform a specified task and may be accessed from anywhere in the program. A long program may be divided into a main section and a series of subroutines. The main section directs the order in which the subroutines do their specialized jobs. This type of organization usually reduces the size of a program because the lines needed to perform a certain task have only to be placed once in a program, even though the task may be performed several times.

GOSUB, RETURN

In the case of programs whose various portions are repeated at different places, it may be efficient to use a subroutine. A subroutine is entered by the statement

```
GOSUB <line number>
```

and exited by the statement

```
RETURN
```

For example:

```
10  ........
20  GOSUB 200
30  ........
40  ........
200 ........
210 ........
220 ........
230 RETURN
999 END
```

The skeleton program illustrates the use of a subroutine. At line 20 the program jumps to the subroutine, which begins at line 200, executes the subroutine and then returns from line 230 to the body of the program at line 30. It is legal to use more than one RETURN statement within a subroutine. You should remember that the computer always returns to the first statement after the GOSUB statement which caused the subroutine to be executed. The major difference between GOSUB and the GOTO statement is that GOSUB permits the use of RETURN. Subroutines are usually placed towards the bottom of a program. They may be written so that one subroutine can access another.

Program 5.1

Given the numbers of pennies, nickels, dimes and quarters as input, this program will output the total amount of money represented by these coins. One subroutine processes and reports the amount of money involved for each of the four types of coins:

```
10 A$ = "PENNIES":V = 1
20  GOSUB 200
30 A$ = "NICKELS":V = 5
40  GOSUB 200
50 A$ = "DIMES":V = 10
60  GOSUB 200
70 A$ = "QUARTERS":V = 25
80  GOSUB 200
90  PRINT "THE TOTAL VALUE WAS $";T / 100
100  END
200  REM  Subroutine to get number of coins
210  PRINT "NUMBER OF ";A$;
220  INPUT N
230  PRINT N;" ";A$;" = ";N * V;" CENTS"
240 T = T + N * V
250  RETURN

]RUN
NUMBER OF PENNIES?4
4 PENNIES = 4 CENTS
NUMBER OF NICKELS?9
9 NICKELS = 45 CENTS
NUMBER OF DIMES?3
3 DIMES = 30 CENTS
NUMBER OF QUARTERS?6
6 QUARTERS = 150 CENTS
THE TOTAL VALUE WAS $2.29

]
```

Line 10 assigns the type (A$) and value (V) of the first coin. Line 20 causes a jump to the subroutine, which begins at line 200, where the number of coins is input. Line 210 prints information about the particular coin being considered, and line 240 adds its contribution to the previous total value of all coins up to this point. Line 250 returns the program to the line following the most recently used GOSUB statement. This entire process, beginning at line 200, repeats itself three more times, and then the dollar total is printed. What would happen if line 100 were deleted?

ON... GOTO

The GOTO statement allows a program to jump to a single specified line. The ON... GOTO statement allows jumps to one of several lines, depending on the value of a numeric variable at the time the statement is encountered. This statement takes the form:

ON <variable> GOTO <list of line numbers>

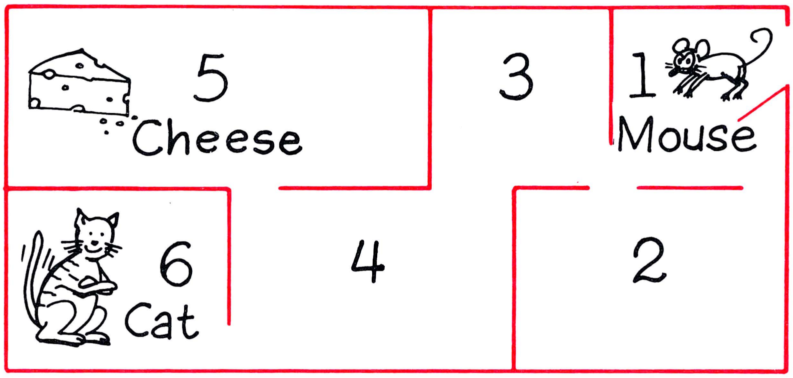
For example:

50 ON X GOTO 100, 120, 140, 160

If X = 1, the program jumps to line 100; if X = 2, it jumps to line 120 and so on. Note that X should not be less than one or greater than the number of line numbers in the list.

Program 5.2

This program simulates the random path taken by a mouse through the following maze:



Once the mouse enters the maze, the door is shut. The mouse wanders about until it either finds the cheese or stumbles upon the cat.

```
100  PRINT 1,
110 X =  INT (2 *  RND (1) + 1)
120  ON X GOTO 200,300
200  PRINT 2,
210 X =  INT (2 *  RND (1) + 1)
220  ON X GOTO 100,300
300  PRINT 3,
310 X =  INT (3 *  RND (1) + 1)
320  ON X GOTO 100,200,400
400  PRINT 4,
410 X =  INT (3 *  RND (1) + 1)
420  ON X GOTO 300,500,600
500  PRINT 5
510  PRINT "THE MOUSE IS GORGING ITSELF!"
520  END
600  PRINT 6
610  PRINT "THE CAT HAS BEEN AWAKENED BY ITS LUNCH."
```

```
]RUN
1               3               1
2               3               2
3               4               5
THE MOUSE IS GORGING ITSELF!

]
```

At line 100 the program prints a 1 to indicate that the mouse is in room #1. Line 110 randomly picks either the number 1 or the number 2. The ON . . . GOTO statement at line 120 sends program execution to line 200 if X = 1, and to 300 if X = 2; lines 200 and 300 correspond to rooms 2 and 3, respectively. Once the mouse is in one of the other rooms, the program prints the room number and decides where the mouse will go from there. Execution continues until the mouse finds itself either in room 5 or room 6.

ON . . . GOSUB

The ON . . . GOSUB statement operates in much the same way as the ON . . . GOTO statement. If there are a number of subroutines in a program, they can be called using the ON . . . GOSUB statement:

```
50 ON X GOSUB 300, 400, 500
```

If X = 1, the program jumps to the subroutine at line 300; if X = 2, to the subroutine at line 400; and if X = 3, to the subroutine at line 500. When the subroutine is completed, the RETURN statement directs the program back to the first statement after line 50.

Program 5.3

This program uses ON . . . GOSUB in delivering a movie review:

```
10  PRINT "1- STAR WARS"
20  PRINT "2- THE EMPIRE STRIKES BACK"
30  PRINT "3- RETURN OF THE JEDI"
40  PRINT "CHOOSE THE MOVIE REVIEW BY NUMBER."
50  INPUT "1,2 OR 3? ";X
60  ON X GOSUB 200,300,400
70  PRINT
80  GOTO 40
200  PRINT "LUKE BATTLES THE EMPIRE"
210  PRINT "RESCUES THE PRINCESS AND"
220  PRINT "DESTROYS THE DREADED DEATHSTAR."
230  RETURN
300  PRINT "THE REBEL ALLIANCE IS TEMPORARILY"
310  PRINT "SETBACK WHEN HAN IS CAPTURED"
320  PRINT "AND FROZEN."
330  RETURN
400  PRINT "YOU'LL HAVE TO SEE IT TO"
410  PRINT "FIND OUT WHAT HAPPENS."
420  RETURN

]RUN
1- STAR WARS
2- THE EMPIRE STRIKES BACK
3- RETURN OF THE JEDI
CHOOSE THE MOVIE REVIEW BY NUMBER.
1,2 OR 3? 1
LUKE BATTLES THE EMPIRE
RESCUES THE PRINCESS AND
DESTROYS THE DREADED DEATHSTAR.

CHOOSE THE MOVIE REVIEW BY NUMBER.
1,2 OR 3?

BREAK IN 50
]
```

Review

1. Write a program in which names are input and then printed. If DONALD is input, use a subroutine to underline the name.

```
]RUN
NAME? MARY
MARY
NAME? SUE
SUE
NAME? DONALD
DONALD
------
NAME?

BREAK IN 10
]
```

2. Write a program to draw five cards from a deck. The computer picks two random numbers, the first between 1 and 4, inclusive, and the second between 1 and 13, inclusive. An ON . . . GOTO statement should use the first number to pick each suit. The second represents the card's value within the suit.

```
]RUN
CARD 1 IS THE 4 OF CLUBS
CARD 2 IS THE 10 OF DIAMOMDS
CARD 3 IS THE 12 OF HEARTS
CARD 4 IS THE 1 OF CLUBS
CARD 5 IS THE 11 OF HEARTS

]
```

3. Write a program using an ON . . . GOSUB statement that takes one of four different courses of action depending on the number entered (A).

If A = 0	Print "You are a zero"
A = 1	Print "Danielle loves Michael"
A = 2	Print "Michael loves Danielle"
A = 3	Print "Who is Danielle?"

Planning a Program

Now that you are familiar with most of the BASIC commands it is important to spend time refining your programming techniques. The first step to this end is learning to plan your program.

The programmer's first concern in planning a program is to define precisely the problem to be solved. This requires the programmer to analyze the problem carefully and to develop a logical sequence of steps to solve the problem. The problem is first broken down into smaller units, each of which should be easier to solve. Just as the task of building a house is made manageable by breaking it down into a series of smaller tasks, so such subdivision of a problem can simplify programming. The solution to the total problem will be accomplished when the solutions to the smaller problems are pieced together in a logical sequence.

The plan used to solve a problem is called an algorithm. Each step in the algorithm should be clear, feasible and unambiguous. A useful technique to develop an algorithm is to determine how a problem could be solved using only pencil and paper. The same step by step sequence required to produce this non-computer solution can usually be translated directly into the sequence employed within a computer program.

Proper planning, as suggested above, will enable the programmer to avoid the common mistake of writing the program before the problem or its solution is understood. Premature "coding" (the term "code" is used to define the instructions which comprise a program) frequently results in a program which must be modified by adding or deleting lines of code until the desired output is achieved, with the consequence that the resulting program is usually a "mish-mash" of statements which do not flow logically from one to another. For example, when excessive GOTO and IF . . . THEN statements have been added to a program to make it work, the sequential flow from line to line, a sign of good programming, is often destroyed. Moreover, the code itself will lack clarity and will be less easily understood by other programmers. Therefore, one should plan a program as thoroughly as possible before approaching the computer. As the computer pioneer R. W. Hamming declared, "Typing is no substitute for thinking."

To demonstrate the use of an algorithm, we will design a program that will give a student a mathematics quiz. This quiz will consist of ten questions, each of which will be one of four types: addition, subtraction, multiplication or division. Two randomly chosen numbers betweeen 1 and 10, inclusive, will be used in each problem, and the student will be given two chances to answer each question correctly. If the student answers a question correctly on the first attempt, then he or she will receive full credit (10 points), or if the second attempt is correct, then half credit (5 points) is awarded. The quiz might be similar to the following:

```
WHAT DOES 5*8 = ? 30
WRONG
WHAT DOES 5*8 = ? 40
CORRECT
WHAT DOES 5 - 12 = ? 7
WRONG
WHAT DOES 5 - 12 = ? -3
WRONG
YOU HAVE HAD 2 TRIES; THE ANSWER IS -7
```

At the outset, a plan or algorithm must be outlined and the problem broken down into separate parts. For example:

(1) Print instructions
(2) Randomly select the type of problem
    (addition, subtraction, multiplication, division)
(3) Choose two random numbers between 1 and 10
    to be used in the problem
(4) Prompt the user for his or her answer
(5) Compare the student's answer to the correct answer
(6) Compute the score
(7) Repeat steps 2 through 6 ten times

Notice that no matter which type of problem (add., subt., mult., or div.) is chosen, the testing procedure remains the same. This means that we could place each type of problem in its own subroutine. Since we want a random selection of four types of problems, we could generate a random number from 1 to 4 and use it as the variable in an ON... GOSUB statement.

At this point it is useful to assign names to the variables we will use and then use these names in refining our algorithm. We will call the variable that represents the type of question (N), the two random numbers used in each question (A) and (B), the student's answer (ANSWER), and the loop variable (QUEST), the correct answer (C), and the score (SCR).

Because ten questions are to be asked, a FOR... NEXT loop will be used. The algorithm using variable names can now be further refined to read:

```
Print Instructions
FOR QUEST = 1 to 10
    Choose random number N from 1 to 4 inclusive
    Choose two random numbers (A,B) from 1 to 10 inclusive
    ON N GOSUB (add, sub., mult., div.)
    FOR GUESS = 1 to 2
        INPUT ANSWER
        Compare ANSWER to correct answer (C)
        Compute SCR
    NEXT GUESS
NEXT QUEST
PRINT SCR
END
Subroutines
Addition
Subtraction
Multiplication
Division
```

The structure of the algorithm is now quite clear. First the student will be given instructions. The body of the program will consist of a main loop which is repeated ten times. Within the loop the computer will choose the type of question to ask and the numbers to be used in the question, ask for an answer, and then compute the score depending on the answer. The last step will print the final score.

Each of the four subroutines will ask one type of problem and compute the correct answer. We have attempted to put parts of the program common to each type of problem in the main body (within the QUEST loop) and only those parts which are unique to a specific type of problem in each subroutine. For example, the subroutine for addition will print the question and determine the correct answer:

```
PRINT A; "+"; B; "= ?"
C = A + B
```

This plan will work for each of the four subroutines, each of which will have a different PRINT statement and answer for the variable C.

In order to structure a program properly, observe the following rules:

1. Place REM statements at the beginning of a program to indicate what the program does and to define the variables used within the program. In addition, REM statements strategically placed throughout the program can be helpful in explaining how the program works.
2. Indent the lines within FOR . . . NEXT loops to clarify where the loops begin and end and what portions of the program are contained within each loop. This is optional, and can be done by inserting colons immediately after the line number, as shown in Program 5.4
3. Break a program down into separate modules or subroutines, where each performs a specific task. It is a good programming practice to separate each subroutine from the others by choosing appropriate line numbers. For example, if a program contains three subroutines, each of which is about 15 lines long, then number the lines of the first subroutine from 100 to 250, the second from 300 to 450, and the third from 500 to 650. In addition, each subroutine should be separated from the next by using blank REM statements, and each subroutine should start with a REM which explains its function.

Program 5.4

With these rules in mind, we are now ready to translate our plan into a program. Note the almost one to one correspondence between the program and the math quiz algorithm:

```
1   REM  N = Type of question
2   REM  ANSWER = Student's answer
3   REM  QUEST = Question number
4   REM  GUESS = Student's attempt number
5   REM  SCR = Student's score
10   PRINT "YOU WILL BE ASKED TEN QUESTIONS THAT"
20   PRINT "COULD BE OF FOUR TYPES: ADDITION,"
30   PRINT "SUBTRACTION, MULTIPLICATION, OR"
40   PRINT "DIVISION.  YOU WILL GET TWO TRIES TO"
50   PRINT "GET EACH CORRECT.  YOUR SCORE WILL"
60   PRINT "BE BASED ON THE NUMBER OF GUESSES"
70   PRINT "YOU TAKE."
80   PRINT
100  FOR QUEST = 1 TO 10
110 ::N =  INT (4 *  RND (1) + 1)
120 ::A =  INT (10 *  RND (1) + 1)
130 ::B =  INT (10 *  RND (1) + 1)
140 :: ON N GOSUB 300,400,500,600
150 :: FOR GUESS = 1 TO 2
160 :::: INPUT "YOUR ANSWER?";ANSWER
170 :::: IF ANSWER <  > C THEN  PRINT "WRONG": GOTO 210
180 :::: PRINT "CORRECT"
190 :::: IF GUESS = 1 THEN SCR = SCR + 10: GOTO 230
200 ::::SCR = SCR + 5: GOTO 230
210 :: NEXT GUESS
220 :: PRINT "YOU'VE HAD TWO TRIES; THE ANSWER IS ";C
230 :: PRINT
240 :: NEXT QUEST
250  PRINT "YOU HAVE SCORED ";SCR;"%"
260  END
270  REM
280  REM
```

```
300  REM   Addition subroutine
310  PRINT A;" + ";B;" = ?"
320 C = A + B
330  RETURN
340  REM
350  REM
400  REM  Subtraction subroutine
410  PRINT A;" - ";B;" = ?"
420 C = A - B
430  RETURN
440  REM
450  REM
500  REM  Multiplication subroutine
510  PRINT A;" * ";B;" = ?"
520 C = A * B
530  RETURN
540  REM
550  REM
600  REM  Division subroutine
610  PRINT A;" / ";B;" = ?"
620 C = A / B
630  RETURN
```

```
]RUN
YOU WILL BE ASKED TEN QUESTIONS THAT
COULD BE OF FOUR TYPES: ADDITION,
SUBTRACTION, MULTIPLICATION, OR
DIVISION.  YOU WILL GET TWO TRIES TO
GET EACH CORRECT.  YOUR SCORE WILL
BE BASED ON THE NUMBER OF GUESSES
YOU TAKE.

8 / 3 = ?
YOUR ANSWER?2.6
WRONG
YOUR ANSWER?2.66
WRONG
YOU'VE HAD TWO TRIES; THE ANSWER IS 2.66666667

5 * 2 = ?
YOUR ANSWER?10
CORRECT

1 + 6 = ?
YOUR ANSWER?7
CORRECT

7 / 6 = ?
YOUR ANSWER?1.167
WRONG
YOUR ANSWER?1.166666
WRONG
YOU'VE HAD TWO TRIES; THE ANSWER IS 1.16666667
```

```
3 + 2 = ?
YOUR ANSWER?5
CORRECT

7 + 6 = ?
YOUR ANSWER?13
CORRECT

8 * 5 = ?
YOUR ANSWER?40
CORRECT

8 + 4 = ?
YOUR ANSWER?12
CORRECT

4 - 3 = ?
YOUR ANSWER?1
CORRECT

7 + 10 = ?
YOUR ANSWER?17
CORRECT

YOU HAVE SCORED 80%

]
```

This math quiz example illustrates that some problems cannot always be foreseen in planning a program. A good algorithm, however, can make correcting these problems substantially easier. In addition, a clearly structured and annotated program can eliminate any unnecessary confusion. The run of Program 5.4 illustrates that there is something wrong in the division subroutine. Although the student answered the division problem correctly each time, the computer did not always acknowledge that fact. The error occurs in the division at line 620, where a rounding error is introduced. At line 170 the student's response, ANSWER, is compared with the desired answer, C. Unfortunately, these variables are compared to ten decimal places of accuracy, and because the student responded by giving an answer to only three decimal places, the comparison failed. The problem might be corrected by rounding the correct answer, C, to 3 decimal places by the line

```
620 C = INT (1000 * A/B + 0.5) /1000
```

but then a new rounding error is introduced by dividing by 1000. In order to get an accurate comparison to only 3 decimal places, it will be necessary to compare the integer values of ANSWER∗1000 with the result multiplied by 1000.

```
170 IF INT (ANSWER*1000 + 0.5) <> INT (C + 0.5)
    THEN PRINT “WRONG”: GOTO 210
```

This statement will correct the rounding problem without affecting the other operations. In addition, the instructions should be modified so that the student is asked to round his or her responses to 3 decimal places.

The following modifications can be made to Program 5.4 to account for the rounding error in the division problems:

```
70   PRINT "YOU TAKE.  ROUND YOUR ANSWERS"
80   PRINT "TO THREE DECIMAL PLACES.  GOOD LUCK!"
90   PRINT
170 :::: IF  INT (ANSWER * 1000 + 0.5) <  > INT (C + 0.5) THEN
       PRINT "WRONG": GOTO 210
220 :: PRINT "YOU'VE HAD TWO TRIES; THE ANSWER IS";C / 1000

320 C = (A + B) * 1000
420 C = (A - B) * 1000
520 C = (A * B) * 1000
620 C =  INT (A / B * 1000 + 0.5)
```

```
]RUN
YOU WILL BE ASKED TEN QUESTIONS THAT
COULD BE OF FOUR TYPES: ADDITION,
SUBTRACTION, MULTIPLICATION, OR
DIVISION.  YOU WILL GET TWO TRIES TO
GET EACH CORRECT.  YOUR SCORE WILL
BE BASED ON THE NUMBER OF GUESSES
YOU TAKE.  ROUND YOUR ANSWERS
TO THREE DECIMAL PLACES.  GOOD LUCK!

5 / 9 = ?
YOUR ANSWER?.55
WRONG
YOUR ANSWER?.555
WRONG
YOU'VE HAD TWO TRIES; THE ANSWER IS .556

     .
     .
     .
```

(A complete run is not shown)

```
4 / 6 = ?
YOUR ANSWER?.667
CORRECT
```

```
5 + 3 = ?
YOUR ANSWER?8
CORRECT

10 - 7 = ?
YOUR ANSWER?3
CORRECT

YOU HAVE SCORED 85%

]
```

Debugging

Debugging is the process of locating errors or "bugs" in a program and then correcting them. Obviously, the best approach is to avoid bugs by carefully planning a program, but even the most carefully written programs will often contain errors. The rounding error encountered in Program 5.4 illustrates how even a well planned program can have unforeseen errors. Two other types of errors which will cause a program to fail are runtime and logic errors.

Runtime Errors

Runtime errors are detected by the computer when a program is run. These errors result from either syntactical and/or semantic errors. Syntactical errors are caused by typing an improper statement. For example,

```
10 REED X, Y
```

should be

```
10 READ X, Y
```

A semantic error is one that violates the computer's rules of operation. For example,

```
10 READ A$, B
20 DATA 35, SMITH
```

is a semantic violation since the computer attempts to assign the string "SMITH" to the numeric variable B. Another common runtime error is caused by using two different variables in defining a FOR . . . NEXT loop:

```
10 FOR X = 1 TO 10
20 PRINT X, X*2, X*3
30 NEXT W
```

Fortunately, the computer detects the runtime error and prints an appropriate error message.

Logic Errors

If the program does not violate any of the syntactical or semantic rules of BASIC, the computer will accept and run it even though the output may not reflect the programmer's true intent. In such a case the errors in the program may stem instead from the programmer's incorrect analysis of how to develop a logical sequence of statements to achieve the intended goal. These forms of errors—referred to as logic errors—are the most difficult to detect and correct.

Program 5.5

This program is supposed to print the areas of circles whose radii are integers varying from 1 to 100 except for those whose computed areas are integers (note line 30). The equation $A = \pi R^2$ is used with the value of π taken as 3.1:

```
10  FOR X = 1 TO 100
20 A = 3.1 * X ^ 2
30  IF A =  INT (A) THEN 10
40  PRINT A,
50  NEXT X
60  END
```

```
]RUN
3.1             12.4            27.9
49.6            77.5            111.6
151.9           198.4           251.1
3.1             12.4            27.9
49.6            77.5            111.6
151.9           198.4           251.1

BREAK IN 40
]
```

Note that the printed areas go as high as 251.1 and then start to repeat. The program actually goes into an infinite loop, which means that it keeps running until the program is halted with CTRL-C.

A good way to detect the logic error in Program 5.5 is to place an extra PRINT statement in the program in order to print the loop variable X. This extra statement can later be removed when the error has been corrected:

25 PRINT “X =”; X,

```
]RUN
X=1             3.1             X=2
12.4            X=3             27.9
X=4             49.6            X=5
77.5            X=6             111.6
X=7             151.9           X=8
198.4           X=9             251.1
X=10            X=1             3.1
X=2             12.4            X=3
27.9            X=4             49.6

BREAK IN 40
]
```

The output indicates that X reaches only 10 and then starts repeating rather than continuing on to 100 as it should. The problem is that line 30 should send the program to line 50 rather than to line 10, where it restarts the X loop at one.

In longer programs it is often useful to place additional print statements at a number of locations to check the value of variables and then remove them when the program is operating properly.

Another technique is to place a line in the program to indicate whether it is getting to a certain point as anticipated. For example:

```
50 PRINT "I'M AT LINE 50 NOW"
```

The line can be removed later when no longer needed.

ONERR GOTO and RESUME

If an error that would stop program execution can be anticipated before a program is run, it is possible to "trap" the error by using the ONERR GOTO statement which allows the program to continue execution after the error has occurred. When an error is encountered, this statement suppresses the printing of an error message and causes program execution to jump to a previously specified line. The general format for the ONERR GOTO statement is:

```
ONERR GOTO <line#>
```

For example,

```
30 ONERR GOTO 60
```

sends program execution to line 60 when an error occurs.

Examples of two frequently encountered errors are that of attempting to divide a number by zero or taking the square root of a negative number. To trap these errors the ONERR GOTO statement must be placed in the program before the error occurs. Then, if an error occurs, program execution will jump to the specified line.

Once an error has been trapped, the statement RESUME will send program execution back to the line where the error occurred. If conditions have not changed, the error will recur and be trapped again, thus making the computer go around in an infinite loop. If conditions are changed by the error trap, program execution will continue from the line where the error had originally occurred.

If an error occurs, the type of error can be identified by examining a special location in the computer's memory. The statement

P = PEEK(222)

assigns the variable P a number corresponding to one of the errors listed in the table below. The PEEK function is described in more detail in Chapter Ten.

Code	Explanation	Code	Explanation
0	NEXT without FOR	120	Redimensioned array
16	Syntax error	133	Division by zero
22	RETURN without GOSUB	163	Type mismatch
42	Out of DATA	176	String too long
53	Illegal quantity	191	Formula too complex
69	Overflow	224	Undefined function
77	Out of memory	254	Bad input
90	Undefined statement	255	CTRL-C typed
107	Bad subscript		

The line number at which an error occurred can be found with the statement

L = PEEK(218) + 256 * PEEK(219)

The variable L is assigned the line number where the error occurred.

The action taken by the ONERR GOTO statement (i.e., jumping to a specified line if an error occurs) can be disabled with the statement

POKE 216,0

After this statement is executed, any errors that occur will cause an error message to be printed and the program will stop running. The POKE statement is used to place a value into the computer's memory and is covered in detail in Chapter Ten.

Program 5.6

This program illustrates how an error handler can determine where and why an error occurrred.

```
10  ONERR  GOTO 100
20 X = 0
30 Y =  - 4
40  PRINT 2164 / X
50  PRINT  SQR (Y)
60  POKE 216,0: REM  disable ONERR GOTO
70  PRINT  LOG ( - 4)
80  END
90  REM
```

```
100  REM  error handler
110 N =  PEEK (222): REM  error number
120 P =  PEEK (218) + 256 *  PEEK (219): REM   error line #
130  PRINT "ERROR ";N;" OCCURRED AT LINE ";P
140  IF N = 133 THEN X = 1
150  IF N = 53 THEN Y =  ABS (Y)
160  RESUME

]RUN
ERROR 133 OCCURRED AT LINE 40
2164
ERROR 53 OCCURRED AT LINE 50
2

?ILLEGAL QUANTITY ERROR IN 70
]
```

Note that when error 133 (division by zero) occurs, the divisor (X) is changed to a non-zero value. When error 53 (illegal quantity) occurs at line 50, the argument (Y) is changed to a non-negative value. Line 60 disables the ONERR GOTO so that when the error occurs in line 70, it is *not* trapped.

Program 5.7

This program finds the quotient of two numbers entered by the user.

```
10  REM  Find the quotient of A and B,
20  REM  which are supplied by the user
30  REM
40  ONERR  GOTO 100
50  INPUT "WHAT ARE A AND B? ";A,B
60 Q = A / B
70  PRINT A;"/";B;" = ";Q
80  PRINT : GOTO 50
100  PRINT A;"/";B;" IS UNDEFINED!"
110  INPUT "NEW A AND B? ";A,B
120  RESUME

]RUN
WHAT ARE A AND B? 3,6
3/6 = .5

WHAT ARE A AND B? 5,0
5/0 IS UNDEFINED!
NEW A AND B? 5,9
5/9 = .555555556

WHAT ARE A AND B?
5/9 IS UNDEFINED!
NEW A AND B?
```

At line 60, the program attempts to divide A by B. If B is zero, an error will occur, since the result of any number divided by zero is undefined. In this event, program execution automatically proceeds to line 100, as instructed by the ONERR GOTO statement at line 40. The program prints its own error message at line 100 and then requests that the user enter new values for A and B. The RESUME statement at line 120 returns program execution to the line where the error occurred, in this case line 60. The program then attempts to find the quotient of the new values. It is important to note that if a new value had not been entered for B, at line 10, the error would have occurred again, sending program execution back to line 100, placing the computer in an infinite loop. Also, the program run was stopped with a CONTROL-C. The error caused by the CONTROL-C would have been trapped by the ONERR GOTO statement, rendering the CONTROL-C ineffective. Instead, the program was stopped with the RESET key.

Hand Tracing

It is almost impossible to guarantee that a program will run properly for all possible situations that it may encounter, but one technique that creates confidence in the validity of a program is called hand tracing. Using test data, the programmer solves calculations by hand and checks the results against those produced by the program. If the computer program produces identical answers for the same data, the programmer is then confident that the program works at least for data that is being tested.

Program 5.8

This is a program which attempts to find the sum of the first N *odd* integers:

```
10  PRINT "NUMBER","SUM"
20 S = 0
30  READ N
40  FOR K = 1 TO N
50 S = S + 2 * K - 1
60  NEXT K
70  PRINT N,S
80  GOTO 30
90  DATA  3,8,10
100  END

]RUN
NUMBER          SUM
3               9
8               73
10              173

?OUT OF DATA ERROR IN 30
]
```

If the output of Program 5.8 is checked by hand for N = 3, S = 1 + 3 + 5 = 9 is a correct answer. In relying on this piece of test data alone, however, the programmer overlooks a serious program error. A major limitation of the test data technique is that the data chosen may bypass the bug. If this program is further hand-tested with N = 8, S = 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 = 64, this result disagrees with the output above. The presence of a bug is now apparent.

The next problem is to locate the position of the error in the program. One effective way to do this, or even better, to avoid errors in the first place, is to "play computer" with the program.

This method is commonly called *hand simulation* or *hand tracing.*

To find the bug in Program 5.8 by this method, the programmer makes a table displaying each variable in the program and then performs the numerical assignments of the variables as the program dictates, line-by-line:

Line	N	K	S	
20			0	
30	3			
40		1		
50			1	
60→40		2		
50			4	
60→40		3		
50			9	
60→70				(Print 3,9)
80→30	8			
40		1		
50			10	(Error! S should now be 1)

The error was caused by not resetting S to 0 before reading the new data N = 8. Hence, to correct the error, change line 80 to:

```
]80 GOTO 20

]RUN
NUMBER           SUM
3                9
8                64
10               100

?OUT OF DATA ERROR IN 30
]
```

TRACE and NOTRACE

The Apple has two commands, TRACE and NOTRACE, that are very useful in aiding a programmer in tracing a program. When the command TRACE is executed, whether in immediate mode or as part of a program line, it instructs the computer to print the line number of every line in a program it executes, as it is executed. This allows a programmer to easily trace the flow of a program. The NOTRACE command cancels the effect of the TRACE command.

Program 5.9

This program prints the combinations of quarters and dimes which add up to $1.00, and demonstrates the effect of the TRACE and NOTRACE commands:

```
1   REM  Q = Quarters
2   REM  D = Dimes
3   REM
10  FOR Q = 0 TO 4
20  FOR D = 0 TO 10
30  IF 25 * Q + 10 * D = 100 THEN
      PRINT "Q=";Q,"D=";D
40  NEXT D
50  NEXT Q
60  END

]TRACE

]RUN
#1 #2 #3 #10 #20 #30 #40 #30 #40 #30 #40
#30 #40 #30 #40 #30 #40 #30 #40 #30 #40
#30 #40 #30 #40 #30 Q=0
D=10
#40 #50 #20 #30 #40 #30 #40 #30 #40 #30
#40 #30 #40 #30 #40 #30 #40 #30 #40 #30
#40 #30 #40 #30 #40 #50 #20 #30 #40 #30
#40 #30 #40 #30 #40 #30 #40 #30 Q=2
D=5
#40 #30 #40 #30 #40 #30 #40 #30 #40 #30
#40 #50 #20 #30 #40 #30 #40 #30 #40 #30
#40 #30 #40 #30 #40 #30 #40 #30 #40 #30
#40 #30 #40 #30 #40 #50 #20 #30 Q=4
D=0
#40 #30 #40 #30 #40 #30 #40 #30 #40 #30
#40 #30 #40 #30 #40 #30 #40 #30 #40 #30
#40 #50 #60
]NOTRACE

]RUN
Q=0             D=10
Q=2             D=5
Q=4             D=0

]
```

Follow the program flow carefully for both loops. Each time a NEXT statement is executed, it returns the program to the statement following the corresponding FOR . . . TO statement.

STOP and CONT

Halting a program at certain points by using one or more STOP statements is a useful debugging technique. Upon interruption, the values of the program's variables may be examined. Typing CONT will continue program execution from the point at which it was interrupted.

Program 5.10

```
10   FOR X = 1 TO 100
20 A = 3 * X ^ 3 + 2 * X ^ 2 + 5
30 B = 7 * X ^ 2 + 2 * X + 5
40   IF A < B THEN  PRINT B;">";A
50   NEXT X
60   END

]RUN
14>10

]
```

When this program is run, the computer prints one solution to the condition at line 40 and then gives the appearance of being at rest until it finally prints the prompt character. Is there only one solution or is there a bug in the program? The values A and B can be checked when X = 50 by adding the following line:

```
45 IF X = 50 THEN STOP
```

The values of A and B can then be examined when the program is halted by typing PRINT A,B. Typing CONT allows this program to continue to completion. Since A is much larger than B when X = 50, it is obvious that there will be no other solutions than the first one. Hence, there is no bug in the program:

```
]RUN
14>10

BREAK IN 45
]PRINT A,B
380005              17605

]CONT

]
```

Review

4. Each of the following programs contains a 'bug'. Find it and correct it.

a)

```
10   INPUT A
20   FOR X = 10 TO 1
30 C = A / X
40   PRINT  INT (C)
50   NEXT X
```

b)

```
10   INPUT N
20   IF N < 0 THEN 10: REM  Prevents negative input
30 P = N * 5
40   IF P < 0 THEN  PRINT "THE PRODUCT IS NEGATIVE": GOTO 60
50   PRINT "THE PRODUCT IS POSITIVE"
60   END
```

Exercises

1. Have the computer produce the printout below. Use a subroutine to print out "PART" and the appropriate number:

```
]RUN

Part 1
****************************************

Part 2
!               !               !

Part 3
ABABABABABABABABABABABABABABABABABABABAB
]
```

2. Write a program that randomly prints one of the following sentences. Using a subroutine, have only the word APPLE flash in each sentence.

AN APPLE A DAY KEEPS THE DOCTOR AWAY.
IS THERE A BUG IN YOUR APPLE?
HOW DO YOU LIKE THEM APPLES?!
DON'T LET YOUR APPLE ROT!

3. Input an integer from 1 to 3. The integer should cause one of the following four sentences to be printed:

FAME IS YOURS IF NO ONE EATS YOUR APPLE.
BLAME IS YOURS IF SOMEONE BEATS YOUR APPLE.
SUCCESS IS YOURS IF YOU DON'T MISTREAT THE APPLE.

use the GET command and put each sentence in its own subroutine.

4. What is the exact output for the following program?

```
10   READ X
20   ON X GOTO 30,40,50,60,70,80,100
30   PRINT "Merrily, ";: GOTO 10
40   PRINT "Row, ": GOTO 10
50   PRINT "Your boat": GOTO 10
60   PRINT "Gently down the stream": GOTO 10
70   PRINT "Life is but a dream": GOTO 10
80   PRINT : GOTO 10
90   DATA  2,2,2,3,4,1,1,1,1,6,5,7
100   END
```

5. (a) Give the exact order in which the lines of the following program are executed:
   (b) If line 80 is changed to:

   80 DATA 3, 0, 4, −5, −1, −8, 1, 5, 999

   what will be the output?

```
10   READ X
20   IF X = 999 THEN 90
30   ON  SGN (X) + 2 GOSUB 50,60,70
40   GOTO 10
50 N = N + 1: RETURN
60 Z = Z + 1: RETURN
70 P = P + 1: RETURN
80   DATA  6,-1,0,999
90   PRINT "N=";N,"Z=";Z,"P=";P
100   END
```

6. (a) Give the exact order in which the lines of the following program are executed:
   (b) If line 90 were changed to:

   90 DATA, 1, 8, −2, 5, −3, −6, 2, 4, −7, 0

   what would the output be?

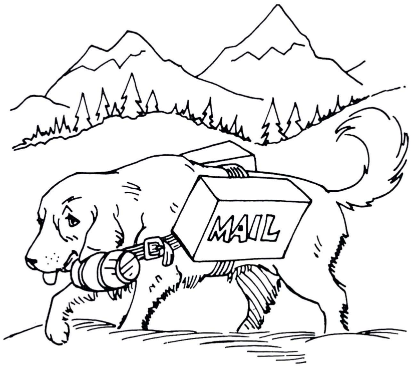
```
10   READ X
20   IF X = 0 THEN 95
30   GOSUB 60
40 T = T + X
50   GOTO 10
60   IF X < 0 THEN 80
70 P = P + X
80   RETURN
90   DATA  5,2,-9,0,3,5
95   PRINT "T=";T,"P=";P
100   END
```

7. Write a program which reads the dimensions of a triangle from a DATA statement and prints its area and perimeter if it is a right triangle. The program should call a subroutine to check whether the triangle is a right triangle or not. Begin by developing an algorithm for this problem.

8. Let X and Y be the coordinates of a point in a plane. Write a program which randomly picks X and Y, each as an integer from -5 and 5, inclusive. The user of the program is to guess X and Y, with the guesses being called X1 and Y1. A subroutine must be used to print the distance between the guessed point and the actual point after each guess. The user is to be given 3 guesses and, if unsuccessful, is then to be given the value of X as well as 2 more tries to guess Y. Begin by developing an algorithm for this problem.

9. Isolated in his ski-lodge in the Swiss Alps, Bjorn Rich often gets his mail late. As a result the statements from his Swiss bank account rarely arrive on time. To solve this problem, write a program to assist Bjorn in keeping his account up to date. Have three subroutines which take care of deposits, withdrawals and the interest (5¾% compounded quarterly). Begin by developing an algorithm for this problem.

```
]RUN
WITHDRAWAL(1),DEPOSIT(2),
CALCULATE INTEREST(3), OR EXIT(4)
1,2,3, OR 4? 2
How much would you like to deposit? 500.75
Your balance stands at $500.75

WITHDRAWAL(1),DEPOSIT(2),
CALCULATE INTEREST(3), OR EXIT(4)
1,2,3, OR 4? 3
How many months since last calculation? 5
Your balance stands at $507.95

WITHDRAWAL(1),DEPOSIT(2),
CALCULATE INTEREST(3), OR EXIT(4)
1,2,3, OR 4? 1
How much would you like to withdraw? 175.45
Your balance stands at $332.5

WITHDRAWAL(1),DEPOSIT(2),
CALCULATE INTEREST(3), OR EXIT(4)
1,2,3, OR 4? 3
How many months since last calculation? 1
Too soon
Your balance stands at $332.5

WITHDRAWAL(1),DEPOSIT(2),
CALCULATE INTEREST(3), OR EXIT(4)
1,2,3, OR 4? 4

]
```



10. The area of any triangle may be found from Hero's formula

$$\text{AREA} = \sqrt{S\,(S-A)\,(S-B)\,(S-C)}$$

where A, B, and C are the length of each side of the triangle and S is the semiperimeter. S = (A+B+C)/2. Write a program that uses Hero's formula in a subroutine to calculate the area of a triangle.

11. Develop an algorithm to play the game Black Jack. The computer will act as dealer in this game. The object of the game is to get closer than your opponent to 21 without going over. Picture cards are worth ten, and aces are worth one or eleven. All other cards are worth their face value. The dealer and the player are each dealt two cards. The player then takes as many cards as he wants to get as close to but not exceed 21. When the player determines that he has enough cards, the dealer takes cards until his cards total at least 17 but not more than 21. The winner is the one who is closest to 21 without going over 21. In the event of a tie the dealer wins.

12. Develop an algorithm for playing the card game War on the computer. The computer will run through the game in a simulation. In this game two players are each given half a normal deck of cards. The top card of each players stack is revealed and the higher card takes the lower card. In the event of a tie each player puts in three cards face down and then reveals a fourth card to determine who will win all ten cards. Whoever runs out of cards first for any reason loses.

13. Each of the following programs contains an error. In each case identify and correct the error(s):

(a)
```
10  READ A,B,C,D
20 E = A * B + C + D
30  PRINT E
40  DATA  2,3,4
50  END
```

(b)
```
10  READ A,B
20  DATA  1,2,3,4
30  PRINT A / B
40  GOTO 20
50  END
```

(c)
```
10  READ A,B,C
20  PRINT A *   / B + C
30  PRINT D / F = ;D / F
40  DATA  5,6,10
50  END
```

(d)
```
10  READ F,G
20  IF F > 5 OR  < 10 THEN 40
30  GOTO 10
40  PRINT F,G
50  DATA  1,10,6,9,11,4
60  END
```

(e)
```
10  FOR X = 1 TO 8
20  FOR Y = 1 TO 3
30 X = X + Y
40  NEXT X
50  NEXT Y
60  PRINT X
70  END
```

(f)
```
10  READ C,D,F
20  DATA  3,6,9,4,7,10
30  PRINT C + D + F
40  GOTO 20
50  END
```

14. Trace the following program by hand and determine its output:

```
10  READ N,A,B
20  FOR I = 1 TO N
30  FOR J = 2 TO N + 1
40 A = 2 * A + B
50 B = 2 * B + A
60  NEXT J
70  PRINT A;"  ";B
80  NEXT I
90  DATA  3,-1,2
100  END
```

15. What output is produced by the following program? (Do not run the program.)

```
10  FOR X = 1 TO 5 STEP 2
20  READ K1,K2
30 A = A + K1 - K2
40 B = B - K1 + K2
50  NEXT X
60  PRINT A,B
70  DATA  1,3,2,4,3,5
80  END
```

16. The following simple programs have errors in logic (i.e. the program runs, but the output is illogical). Find and correct the errors:

(a)
```
10  READ A,B,C
20  PRINT A + B + C
30  GOTO 20
40  DATA  1,2,3,4,5,6
50  END
```

(b)
```
10  FOR X = 1 TO 10
20  IF X > 5 THEN 50
30  PRINT X;" is greater than 5"
40  GOTO 60
50  PRINT X;" is less than 5"
60  NEXT X
70  END
```

```
(c) 10  READ A,B,C
    20  FOR X = 1 TO 10
    30 Y = (A * B) / (C - X)
    40  IF Y < 1 THEN 50
    50  PRINT "Y<1";Y
    60  NEXT X
    70  DATA  20,10,5,5,10,20
    80  END
```

17. Each of the following programs contains errors which will result in error messages being printed by the computer when the program is run. Find and correct the errors:

```
(a) 10  PRINT "What is the formula weight of element"
    20  INPUT "Formula weight"
    30  IF X > 20 THEN 50
    40 X = X / 2
    50  PRINT X
    999  END
```

```
(b). 15  FOR X = 1 TO 2
     25  FOR Y = 1 TO 3
     35  IF Y = X THEN A = A + 1: GOTO 50
     45  NEXT Y
     55  PRINT "X and Y are equal ";A;"times"
     65  NEXT X
     75  END
```

```
(c) 5  READ B,A,C
    10 X1 = (B + ( SQR (B ^ 2 - 4 * A * C) / 2 * A)
    20 X2 = (B + ( SQR (B ^ 2 - 4 * A * C) / 2 * A)
    30  IF X1 > X2THE50
    40  GOTO 5
    50  PRINT "X1=";X1;"X2=";X2
    60  GOTO 5
    70  DATA  1,2,3,4,5,6
    80  END
```

```
(d) 10  READ A,B,C
    20 X = A * B + C
    30  IF X > 200 THEN LINE10
    40  PRINT X
    50  DATA  25,26,27,28,29,30
    60  END
```

```
(e) 10  FOR X = 1 TO 26
    20 A$ = A$ + X
    30  NEXT X
    40  PRINT A$
    50  END
```

18. The following program is designed to arrange sets of numbers in descending order. If the second number is larger than the first, the computer interchanges the two values. (This occurs in lines 30 and 40.) Explain the output and correct the program to give the desired output:

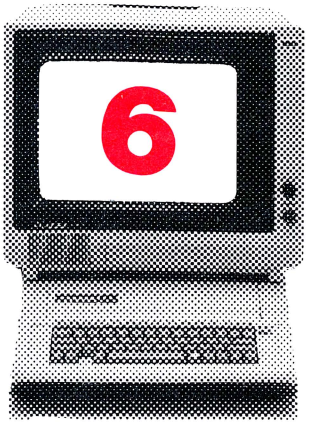
```
10   READ A,B
20   IF A > B THEN 50
30 A = B
40 B = A
50   PRINT A,B
60   GOTO 10
70   DATA  10,20,20,10
80   END

]RUN
20               20
20               10

?OUT OF DATA ERROR IN 10
]
```

19. The area of a square that measures one inch on a side is equal to one square inch. If the computer takes various slices of this same square and sums the areas of the slices, the answer should be one square inch. In the following program 100, 500, 1000 and 1024 slices are used and the output printed below. Explain the results:

```
10   READ N
20   FOR X = 1 TO N
30 Y = Y + 1 / N
40   NEXT X
50   PRINT Y;" sq. inches for ";N;" slices"
60 Y = 0
70   GOTO 10
80   DATA  100,500,1000,1024
90   END

]RUN
1 sq. inches for 100 slices
1.00000002 sq. inches for 500 slices
.999999975 sq. inches for 1000 slices
1 sq. inches for 1024 slices

?OUT OF DATA ERROR IN 10
]
```

20. Write a program that uses the Quadratic Formula: $\dfrac{-b \pm \sqrt{b^2-4ac}}{2a}$ to find the roots of an equation. Input a, b, c and have the computer print the two roots. Your program should have an error trap for the error that might occur when the computer attempts to take the square root of a negative number. When this situation arises have the computer print "roots are imaginary."

21. Write a program that prints all the results of 10 divided by all integers between −5 and 5 inclusive. Your program should trap the error that occurs when you try calculating 10/0.



GRAPHICS

GR

TEXT

HOME

COLOR

PLOT

HLIN

VLIN

SCRN

HGR

HGR2

HCOLOR

HPLOT

This chapter introduces the statements needed for producing graphics and shows how graphics programs use them. The Apple provides two graphics modes: low and high resolution. The low resolution mode divides the graphing area into 40 rows, each which is further subdivided into 40 brick-shaped areas. Each brick represents a point and may be filled with any of 16 colors. The high resolution mode divides the graphing area into very small dots which may be plotted in 8 different colors. Statements are available for plotting lines and points in both modes. Low resolution mode is usually used to produce bar graphs or simple figures, while the high resolution mode is used to graph mathematical functions or pictures where greater detail and higher precision are desired.

If the Apple is used with a black and white video display, the colors referred to in this chapter will be displayed as various shades of gray and black so the description of the color variables is still useful.

Low Resolution Graphics

The low resolution mode partitions the screen into an invisible 40 by 40 grid and gives the user a selection of 16 colors which can be employed to fill a rectangle in the grid. The valid X and Y values of the grid range between 0 and 39, inclusive, where X is the horizontal coordinate and Y the vertical coordinate. The origin (0,0) of the grid is located in the upper-left corner of the display screen. The grid occupies the top portion of the screen, while the bottom 4 lines of the screen contain a small "text window" for program lines and character output. All of the graphing statements discussed may be used either in immediate mode or in a program.

GR

In order to enter the low resolution mode, the programmer must employ the GR statement either in immediate mode or in a program. GR will clear the 40 × 40 grid on the user's video display in preparation for plotting. For example,

```
20 GR
```

erases the video screen and prepares the computer to accept low resolution statements.

TEXT and HOME

In order to leave either of the two graphics modes and return to normal programming mode, the TEXT statement is used. If the line

```
40 TEXT
```

is given from low resolution graphics mode, the computer is reset to normal mode, but the screen is not completely cleared. The low resolution graphics currently on the screen will be lost and the screen will be filled with miscellaneous characters and text.

To clear the screen the HOME command is used. For example,

50 HOME

will clear the screen and put the cursor in the upper left hand corner of the screen. If the HOME statement is used while in low resolution graphics mode, then only the bottom four lines (the text window) are cleared. The graphics are unaffected.

COLOR

When plotting points and lines in low resolution mode, the programmer can use the COLOR variable. A value between 0 and 15 is assigned to this variable to select one of 16 colors. The COLOR variable is set like any other variable. For example, the line

120 COLOR = 9

will set the value of the COLOR variable to 9. Thereafter, all points plotted will be orange until the value of COLOR is changed. The COLOR variable is set to zero automatically when the GR statement is used. The valid colors are:

0—Black
1—Red
2—Dark Blue
3—Purple
4—Dark Green
5—Gray
6—Medium Blue
7—Light Blue
8—Brown
9—Orange
10—Gray
11—Pink
12—Green
13—Yellow
14—Aqua
15—White

PLOT

In low resolution plotting the PLOT statement is used to plot a single-brick-shaped area on the display screen in the 40×40 grid. The color of the brick is determined by the value of the COLOR variable. The statement

20 PLOT X,Y

will plot a brick at the coordinates (X,Y). Remember that the values for X and Y range from 0 to 39.

Program 6.1

The following program will plot 3 red bricks on the display screen at the points (12,32), (29,11), and (1,2):

```
10  GR
20  COLOR= 1
30  PLOT 12,32
40  PLOT 29,11
50  PLOT 1,2
```

HLIN

The HLIN statement is used to plot a low resolution horizontal line of bricks. The statement

45 HLIN X1,X2 AT Y

will plot a horizontal line from point (X1,Y) to (X2,Y). The color of the line will be the last one set by the COLOR variable before the HLIN statement is executed. For example,

30 HLIN 10,30 AT 22

will draw a line from coordinates (10,22) to (30,22). If COLOR = 9 when line 30 is executed, then the line will be orange.

VLIN

The VLIN statement operates similarly to the HLIN statement but produces a vertical line. The statement

110 VLIN Y1, Y2 AT X

will plot a vertical line from point (X,Y1) to (X,Y2). The color of the line will depend upon the last setting of the COLOR variable. For example,

180 VLIN 0,39 AT 21

will draw a vertical line from coordinates (21,0) to (21,39). Assuming that COLOR = 1 when line 180 is executed, a red line will be produced.

Program 6.2

This program demonstrates the various statements used in low resolution graphing:

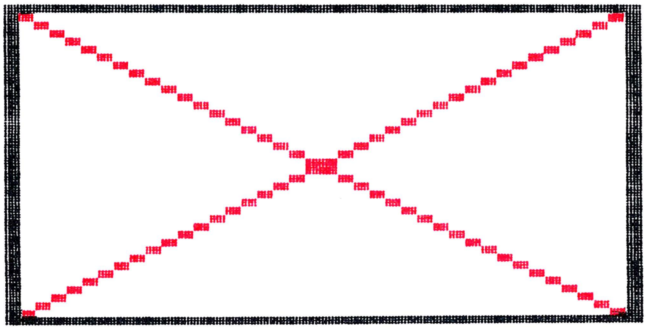
```
10  GR
20  COLOR= 1
30  PLOT 20,12
40  HLIN 0,39 AT 30
50  COLOR= 15
60  HLIN 12,32 AT 32
70  VLIN 1,18 AT 24
```

Line 10 prepares the display screen for graphing, while line 20 selects the color red. The plotting done by lines 30 and 40 wil be in red. Line 30 graphs a point at (20,12), while line 40 draws a horizontal line from (0,30) to (39,30). Line 50 changes the color to white so that lines 60 and 70 will produce white lines, but what was plotted by lines 30 and 40 will remain red.

Program 6.3

This program uses low resolution graphics to draw a large red X (using PLOT) with a green border around the screen (using HLIN and VLIN).

```
10   GR
20   COLOR= 12
30   HLIN 0,39 AT 0: HLIN 0,39 AT 39
40   VLIN 0,39 AT 0: VLIN 0,39 AT 39
50   REM
100   REM  now draw the red "X"
110   COLOR= 1
120   FOR I = 1 TO 38
130   PLOT I,I
140   PLOT 39 - I,I
150   NEXT I

]RUN
```

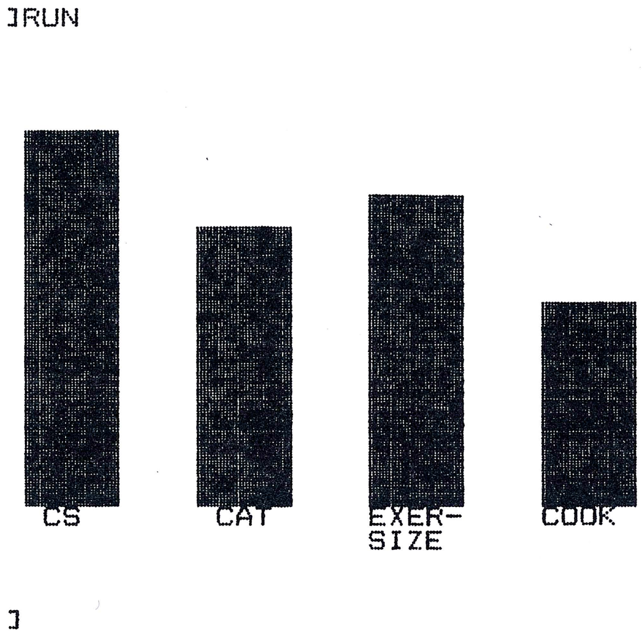


Line 10 prepares the screen for low resolution graphics. Then line 20 selects green as the color for the border which is drawn by lines 30 and 40. Line 30 draws the upper and lower portions of the border, while line 40 draws the right and left sides. Next, line 110 selects the color red and the loop from 120-150, which contains two PLOT statements, draws the X. The PLOT at line 130 draws a diagonal from the upper left of the screen to the lower right, one brick at a time from positions (1,1), (2,2), . . . up to (38,38). Similarly, the PLOT at line 140 draws a diagonal from the upper right to the lower left.

Program 6.4

This program uses low resolution graphics commands to produce a bar graph for Bob's Bookstore. The data to be graphed represents the sales of four different books over a six day period:

```
1   REM  CS = sales of computer science books for one day
2   REM  CT = sales of cat books for one day
3   REM  EX = sales of exercise books for one day
4   REM  CK = sales of cookbooks for one day
5   REM  B()= total sales for the week of each book
6   REM  BK = book number
7   REM  C = column number for a specific book
8   REM
10  REM  read number of books sold
20  FOR D = 1 TO 6
30  READ CS,CT,EX,CK
40 B(1) = B(1) + CS:B(2) = B(2) + CT
50 B(3) = B(3) + EX:B(4) = B(4) + CK
60  NEXT D
70  REM
100  REM  draw bar graph
110  GR : COLOR= 15
120  FOR BK = 1 TO 4
130  FOR C = 1 TO 5
140  VLIN 40 - B(BK),39 AT (BK - 1) * 9 + C
150  NEXT C
160  NEXT BK
170  REM
200  REM  label each bar
210  PRINT  TAB( 3);"CS"; TAB( 12);"CAT"; TAB( 20);"EXER-";
220  PRINT  TAB( 29);"COOK"
230  PRINT  TAB( 20);"SIZE"
240  REM
300  REM  data section
310  DATA  8,10,3,6
320  DATA  4,2,4,5
330  DATA  3,7,6,2
340  DATA  9,1,3,3
350  DATA  7,6,5,1
360  DATA  4,0,8,2
```



```
]RUN

  CS       CAT     EXER-    COOK
                   SIZE

]
```

The loop from lines 20-60 reads in and totals the number of computer science, cat, exercise and cookbooks sold during the week. Line 110 sets up low resolution graphics mode with the color white. Lines 120-160 make the bars that consist of 5 vertical columns drawn side by side to the proper height for each of the four books. The major work is done in line 140. The number of books sold is used to determine the starting position of the columns to be drawn for a specific book. For example, if 10 books were sold, the columns would go from 30 to 39, which is a height of 10. This is done since the last row on the screen is 39, which is the position the bars are drawn to. Next, the book number (BK) and column number (C) is used to determine where the vertical columns will be drawn. The expression (BK-1)*9 puts a separation of 9 spaces from the left hand side of one bar to the left hand side of the next bar. When C is added to this with values of 1 to 5, it causes 5 columns to be drawn side by side to make the bar for each book. This leaves 4 spaces between the right hand side and the left hand side of consecutive bars.

SCRN

The SCRN function makes it possible for a program to determine the color of a particular block in the 40 × 40 grid. Given coordinates (X,Y), the SCRN function will return a value between 0 and 15 corresponding to the color value at the given point. The coordinates X and Y must be between 0 and 39, inclusive. The following statements illustrate various uses of the SCRN function:

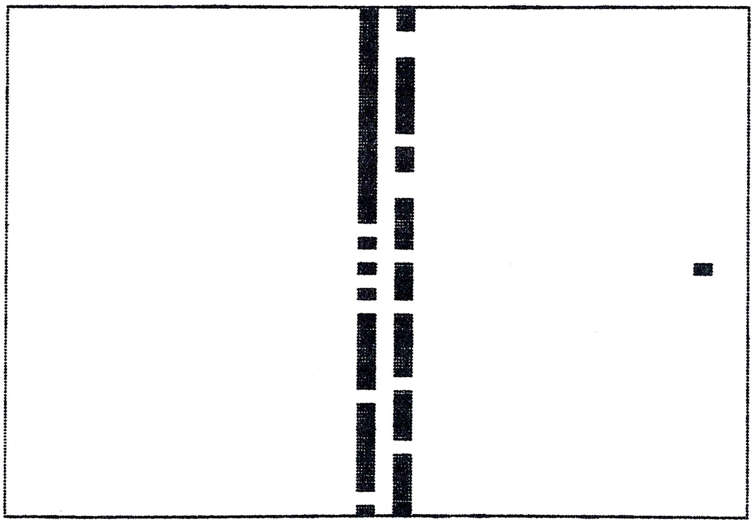
```
10 C = SCRN(10,32)
80 PRINT "THE COLOR IS"; SCRN(18,Y2)
45 PRINT 16*SCRN(18,2)–SCRN(I,J)
```

SCRN can be used in immediate mode by typing statements in the text window.

Program 6.5

This program uses low resolution graphics to place two "walls" down the center of the screen. It then places a "ball" in motion starting at the upper left hand corner of the screen. This ball will travel in a straight line until one of two things occur. First, if the ball hits a wall, it ricochets off, leaving a hole where it hit. Second, if it hits any of the four sides of the screen, it ricochets:

```
1   REM  X,Y = coordinates of the ball
2   REM  XN = the new X coordinate
3   REM  YN = the new Y coordinate
4   REM  XD = the X direction of motion
5   REM  YD = the Y direction of motion (for each space in
      the X direction)
6   REM
10   GR : COLOR= 3
20   VLIN 0,39 AT 19: VLIN 0,39 AT 21
30  X = 0:Y = 0
40  XD = 1:YD = 1
50   COLOR= 5: PLOT X,Y
60  XN = X + XD
70   IF XN = 0 THEN XD = 1
80   IF XN = 39 THEN XD =  - 1
90  YN = Y + YD
100   IF YN < 0 THEN YD =  INT (2 *  RND (1) + 1):YN = Y +
      YD: GOTO 100
110   IF YN > 39 THEN YD =  -  INT (2 *  RND (1) + 1):YN =
      Y + YD: GOTO 110
120   IF  SCRN( XN,YN) = 3 THEN XD =  - 1 * XD
130   COLOR= 0: PLOT X,Y
140  X = XN:Y = YN
150   GOTO 50
```

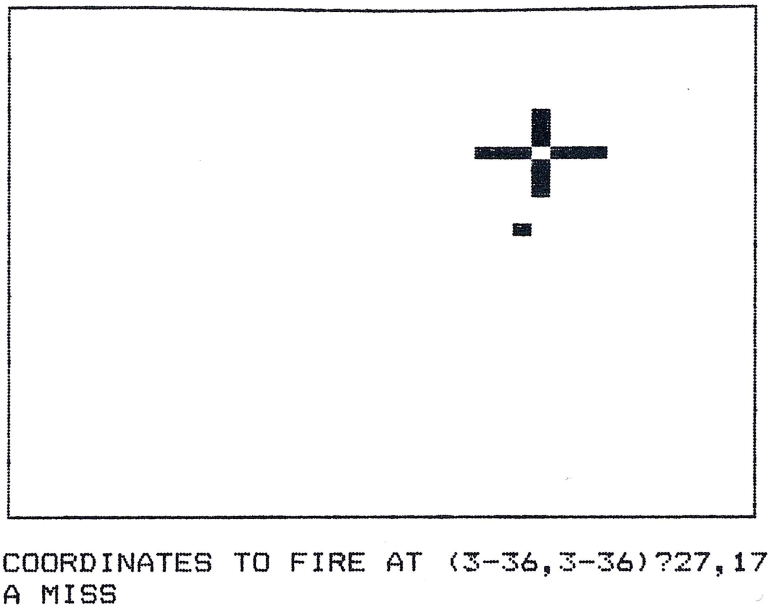


Line 10 sets up the low resolution graphics mode with the color purple chosen for the two "walls" that are drawn by line 20. Line 30 sets up the ball's initial coordinates (X,Y), and line 40 sets up the initial X direction (XD) and Y direction (YD) of the ball, causing the ball to move 1 space in the +X direction for every 1 space in the +Y direction. The ball is then drawn in gray by line 50. The new X coordinate (XN) is determined by line 60, while lines 70 and 80 modify the X direction if the ball has hit the left or right edge of the screen. Line 90 evaluates the new Y coordinate (YN), and lines 100-110 modify the Y direction if the top or bottom edge of the screen is hit, randomly moving the ball one or two spaces in the Y direction for each space it moves in the X direction. This is done to prevent the ball from getting into an infinitely repeating pattern. Line 120 uses SCRN to check if the ball has hit one of the two walls by checking for the color purple, in which case it reverses the X direction. Line 130 continually erases the old position of the ball. If the ball has hit a brick in the wall, this line also erases the brick since it is at the same position as the ball. Finally, X and Y, the coordinates of the ball, are now set to the new coordinates, and the entire process is repeated.

Program 6.6

The following game program simulates a one-on-one dogfight between you and the computer. The mission is for you to align the viewsight directly on a target randomly placed by the computer. If you are successful, the enemy is destroyed. However, if you miss, the enemy may move one space in one of eight directions before you are able to fire again.

```
1  REM  XP,YP = coordinates the player is shooting at
2  REM  XE,YE = coordinates of the enemy
3  REM  X1,Y1 = random integers (-1 to 1) to move the enem
     y after a missed shot
4  REM
10  GR
20  REM  draw enemy at random initial coordinates
30 XE =  INT (34 *  RND (1) + 3):YE =  INT (34 *  RND (1) +
     3)
40  COLOR= 3: PLOT XE,YE
50  INPUT "COORDINATES TO FIRE AT (3-36,3-36)?";XP,YP
60  REM  check for improper input
70  IF XP < 3 OR XP > 36 OR YP < 3 OR YP > 36 THEN 50
80  REM
100  REM  draw crosshairs
110  COLOR= 5
120  HLIN XP - 3,XP + 3 AT YP
130  VLIN YP - 3,YP + 3 AT XP
140  COLOR= 0: PLOT XP,YP
150  FOR D = 1 TO 200: NEXT D: REM  delay loop to display
     crosshairs
160  REM
200  REM  check for a win, remove old figures
210  IF XP = XE AND YP = YE THEN 400
220  PRINT "A MISS"
230  HLIN XP - 3,XP + 3 AT YP
240  VLIN YP - 3,YP + 3 AT XP
250  PLOT XE,YE
260  REM
300  REM  move enemy one space in any direction
310 X1 =  INT (3 *  RND (1) - 1):XE = X1 + XE
320  IF XE < 3 OR XE > 36 THEN XE = XE - X1: GOTO 310
330 Y1 =  INT (3 *  RND (1) - 1):YE = Y1 + YE
340  IF YE < 3 OR YE > 36 THEN YE = YE - Y1: GOTO 330
350  GOTO 40
400  PRINT "A HIT, YOU WIN!!"
```



```
COORDINATES TO FIRE AT (3-36,3-36)?27,17
A MISS
```

Line 50 allows the user to input the coordinates to be fired upon (XP,YP), and lines 110-140 draws and centers the crosshairs on this coordinate. The viewsight and the enemy are erased by lines 230-250 if the player misses, and the enemy's new coordinates are randomly calculated by lines 310-340. This is done by picking 2 random numbers from −1 to 1 and adding them to the old X and Y coordinates of the enemy. Lines 320 and 340 guarantee that the enemy remains on the screen.

Review

1. Have the computer draw a green border around the edge of the graphing area. Make the border 1 'brick' wide.
2. Have the computer color the screen red except for a single blue point at coordinates (19,19).
3. Have the computer generate 15 vertical lines, one in each color except black.

High Resolution Graphics

High resolution graphics enables the user to draw more finely detailed images than those drawn in low resolution graphics, because the display screen is partitioned into a grid of dots which are smaller and far more numerous than the bricks used in low resolution graphics. There are two modes used in high resolution graphics. One mode utilizes the entire screen for a display grid of 192 rows by 280 columns. The other mode reserves a 4 line text window, which reduces the grid to 160 rows by 280 columns. As in low resolution graphics, the origin (0,0) is located in the upper-left corner of the screen.

HGR and HGR2

The HGR and HGR2 statements are used to switch the display screen into high resolution mode. The HGR statement reserves a text window at the bottom of the screen, while the HGR2 statement utilizes the entire screen for graphing. Both statements clear the high resolution graphing area. Valid examples are:

```
70 HGR2
```

and

```
45 HGR
```

HCOLOR

When plotting in high resolution graphics use the HCOLOR variable for color selection. A range of 8 colors is available, numbered 0 through 7, inclusive. For example, the statement

```
139 HCOLOR = 1
```

will set the value of the HCOLOR variable to 1, thus selecting the color green. All subsequent plotting will be in green until the HCOLOR variable is changed. The valid colors for high resolution plotting are:

0—Black
1—Green
2—Purple
3—White
4—Black
5—Red
6—Blue
7—White 2

Because of variations in colors produced by different video displays, the colors produced may not agree exactly with those in the table.

HPLOT

The HPLOT statement is used for plotting both points and lines in high resolution graphics. The statement

```
20 HPLOT X,Y
```

will plot a single point at the coordinates (X,Y). The X-coordinate can vary from 0 to 279. If the 4 line text window is used, then the Y-coordinate can vary from 0 to 159; otherwise, Y can vary from 0 to 191. The HPLOT statement can be modified to draw lines. The statement

```
180 HPLOT X1,Y1 TO X2,Y2
```

draws a line from the point (X1,Y1) to (X2,Y2) with its color determined by the HCOLOR variable. Several connecting lines can be drawn in a single statement. For example, the statement

```
210 HPLOT X1,Y1 to X2,Y2 to X3,Y3
```

draws one line from (X1,Y1) to (X2,Y2) and another line from (X2,Y2) to (X3,Y3). The modifier TO can be added to the HPLOT statement as many times as needed except for the restriction that a program line is limited to 239 characters.

If the first set of coordinates after the word HPLOT is omitted, a line is drawn drom the last point plotted by a previous HPLOT statement to the point specified after the word TO. The color of the new line will be that of the last point plotted and not the color corresponding to the most recent value of the HCOLOR variable. An example of this form of the HPLOT statement is:

```
40 HPLOT TO X,Y
```

Because of the way the Apple displays colors, there may be some difficulty with the color of points and vertical lines in high resolution. The difficulty may be avoided by plotting even numbered colors at points with even numbered X-coordinates, and odd numbered colors at points with odd numbered X-coordinates. For example, to plot a red (COLOR = 5) line with an X-coordinate of 21, use the statement

```
20 HPLOT 21,3 TO 21,12
```

Should we attempt to plot a red line with an X-coordinate of 22, the statement

```
20 HPLOT 22,3 TO 22,12
```

will plot a black line which cannot be seen against the screen's black background.

The only exception to this rule is the color white, (HCOLOR = 3 or HCOLOR = 7). It is impossible to draw a single white vertical line or a single white point on the screen. To generate white vertical lines draw 2 vertical lines next to each other with HCOLOR = 3 or HCOLOR = 7.

Also, care must be taken to avoid drawing two differently colored vertical lines next to each other on the screen, since the resulting color will rarely be as intended.

Program 6.7

The following program, using high resolution graphing statements, constructs a green border around the display area and plots a single blue point:

```
10  HGR
20  HCOLOR= 1
30  HPLOT 1,1 TO 279,1 TO 279,159
40  HPLOT  TO 1,159
50  HPLOT  TO 1,1
60  HCOLOR= 6
70  HPLOT 140,80
```

Lines 30 through 50 draw the green border around the edge of the display screen. Note that the last point plotted before line 40 was (279,159) at line 30. Therefore, line 40 has the same effect as the statement

```
40 HPLOT 279,159 TO 1,159
```

A similar comment applies at line 50. Line 70 plots a single blue point in the center of the display. Note that if line 30 is changed to

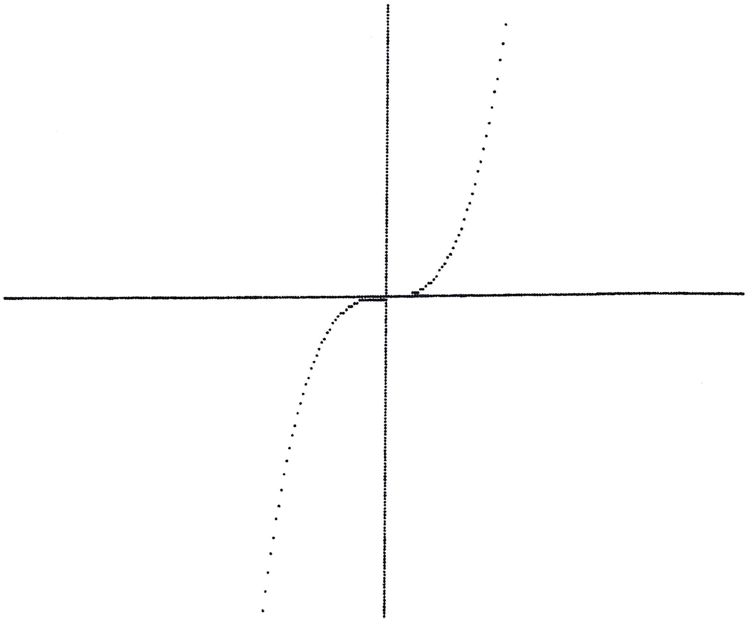
30 HPLOT 1,1 TO 278,1 TO 278,159

the right vertical side of the border might be black and thus not visible since an attempt has been made to plot an odd numbered color with an even numbered X-coordinate.

Program 6.8

The following program plots the function (X ↑ 3)/1000. The axes are conventional, the positive X-axis pointing to the right and the positive Y-axis pointing upward:

```
1   REM   X0,Y0 = coordinate of origin
2   REM   X = value to be used in the equation
3   REM   Y = result of the equation
4   REM   X1,Y1 = coordinates to be plotted on HI-RES screen
5   REM
10   HGR2 : HCOLOR= 7
20  X0 = 135:Y0 = 100
30   REM  draw axes
40   HPLOT 0,Y0 TO 279,Y0
50   HPLOT X0,0 TO X0,191
60   REM
100   REM  graph function
110   FOR X1 = 0 TO 279
120  X = X1 - X0
130  Y = (X ^ 3) / 1000
140  Y1 = Y0 - Y
150   IF (Y1 >  = 0) AND (Y1 <  = 191) THEN  HPLOT X1,Y1
160   NEXT X1

]RUN
```



Line 10 prepares the display for graphing and selects the color white 2 while line 20 places the origin (intersection of X and Y axes) at (X0,Y0). At this point on the screen, both X and Y are zero. The actual plotting of the axes is done by lines 40 and 50. A loop is established at line 110 where values for X1 are generated, and X calculated by subtracting X0 from X1. This is subtracted since we want to use negative numbers in the equation when plotting is done to the left of the origin. Y is calculated from the desired equation at line 130. At line 140 Y1 is calculated similarly to X1 in preparation for plotting the point. Line 150 determines whether the values of X1 and Y1 fit into the graphing area. If so, the point (X1,Y1) is plotted.

Review

4. Allow the user to input two values, X and Y. Have the computer then take these two values and plot a point at the given coordinates (X,Y) with a randomly selected color.

5. Have the computer generate horizontal lines, one for each possible color of high resolution graphics (except HCOLOR = 0). Each color should be separated by a black area 10 lines wide.

6. Modify Program 6.8 so that it will plot the equation $Y = (1/5)X^2 - 8$.

Switching Displays

The Apple has two sections of high resolution graphics memory: HGR and HGR2. It is possible to save pictures in either or both of these sections and then to display them without clearing the screen. This is done with various POKE statements. A POKE statement places information in a specified memory location. At present it is not important to understand exactly how POKE works. Simply follow the commands given to get the display desired. For more information regarding POKE see Chapter Ten.

If TEXT mode is employed after HGR has been used to draw a picture, the picture may be redisplayed by use of the statement:

```
POKE -16304,0
```

If TEXT mode is used after HGR2 has been employed to draw a picture, the user may redisplay the picture by using the statements:

```
POKE -16304,0 : POKE -16299,0
```

The first POKE causes HGR to be displayed, and the second POKE is the command to switch from HGR to HGR2. It is also possible to switch from HGR2 to HGR. This is done with the statement:

```
POKE -16300,0
```

Remember that the use of TEXT mode does not erase the contents of the high resolution screen memory even though it is no longer displayed.

Normally, HGR, high resolution graphics section 1, has a 4 line text window at the bottom of it, and HGR2, high resolution graphics section 2, has all graphics. It is possible to remove the text window from section 1 (HGR) and display all graphics with the statement:

```
POKE -16302,0
```

To put the text window back, use the statement:

POKE −16301,0

These statements may also be used in section 2; however, the text window displayed is not easily accessible to the user. Therefore, it is recommended that HGR2 be used only in full display graphics mode.

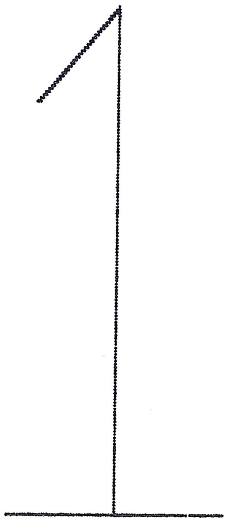
Program 6.9

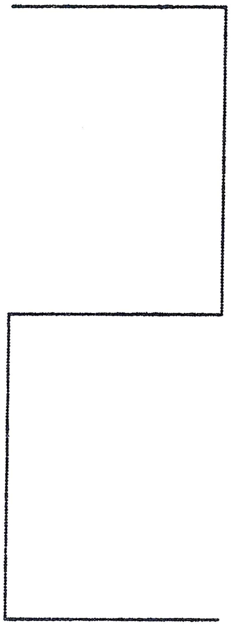
This program demonstrates how to switch between the different display modes without clearing the screen. A message is written in TEXT, a large figure "1" is drawn in HGR, and a large figure "2" is drawn in HGR2. These are then displayed one at a time according to the input of the user. Pressing a "T" causes TEXT to be displayed, pressing a "1" causes HGR to be displayed, and pressing a "2" causes HGR2 to be displayed:

```
10  REM  put a message in the middle of the TEXT display
20  TEXT
30  HOME
40  VTAB 12
50  PRINT  TAB( 14);"THIS IS TEXT"
60  REM
100  REM  draw a large 1 in HGR display
110  HGR
120  HCOLOR= 7
130  HPLOT 110,30 TO 140,0 TO 140,159
140  HPLOT 100,159 TO 180,159
150  REM
200  REM  draw a large 2 in HGR2 display
210  HGR2
220  HCOLOR= 7
230  HPLOT 100,0 TO 180,0 TO 180,96
240  HPLOT  TO 100,96 TO 100,191 TO 180,191
250  REM
300  REM  get input from user to decide which screen is to
      be displayed
310  GET D$
320  IF D$ = "T" THEN  TEXT : GOTO 310
330  REM
400  REM  if a 1 was pressed display HGR
410  IF D$ <  > "1" THEN 500
420  POKE  - 16304,0: REM  enter graphics
430  POKE  - 16301,0: REM  keep 4 line TEXT window
440  POKE  - 16300,0: REM  switch to HGR if HGR2 was the l
     ast screen displayed
450  GOTO 310
460  REM
500  REM  if a 2 was pressed display HGR2
510  IF D$ <  > "2" THEN 310: REM  invalid input
520  POKE  - 16304,0: REM  enter graphics
530  POKE  - 16302,0: REM  make the display all graphics
540  POKE  - 16299,0: REM  switch to HGR2
550  GOTO 310
```

Each of these figures appears separately on the display screen when Program 6.9 is run:

```
THIS IS TEXT
```



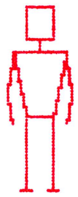


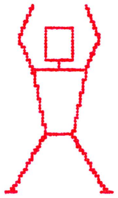
Program 6.10

This program creates some simple high resolution animation. A figure is drawn in HGR and another figure is drawn in HGR2. By alternating which display is being shown, the figure can appear to be moving, and in this case it appears to be doing jumping jacks:

```
10   REM  draw first picture in HGR
20   HGR : HCOLOR= 5: POKE  - 16302,0: REM  full page graph
      ics
30   HPLOT 135,91 TO 145,91 TO 145,101 TO 135,101 TO 135,91
      : REM  head
40   HPLOT 140,101 TO 140,105: REM  neck
50   HPLOT 130,105 TO 150,105 TO 145,125 TO 135,125 TO 130,
      105: REM  body
60   HPLOT 130,105 TO 128,116 TO 130,127: REM  left arm
70   HPLOT 150,105 TO 152,116 TO 150,127: REM  right arm
80   HPLOT 135,125 TO 135,148: REM  left leg
90   HPLOT 145,125 TO 145,148: REM  right leg
100  HPLOT 133,148 TO 137,148: REM  left foot
110  HPLOT 143,148 TO 147,148: REM  right foot
120  REM
200  REM  draw second picture in HGR2
210  HGR2
220  HPLOT 135,97 TO 145,97 TO 145,107 TO 135,107 TO 135,9
      7: REM  head
230  HPLOT 140,107 TO 140,110: REM  neck
240  HPLOT 130,110 TO 150,110 TO 145,130 TO 135,130 TO 130
      ,110: REM  body
```

```
250  HPLOT 130,110 TO 125,101 TO 130,90: REM  left arm
260  HPLOT 150,110 TO 155,101 TO 150,90: REM  right arm
270  HPLOT 135,130 TO 122,148: REM  left leg
280  HPLOT 145,130 TO 158,148: REM  right leg
290  HPLOT 120,148 TO 124,148: REM  left foot
300  HPLOT 156,148 TO 160,148: REM  right foot
310  REM
400  REM  alternately display the pictures
410  POKE  - 16300,0: REM   display HGR
420  FOR D = 1 TO 200: NEXT D
430  POKE  - 16299,0: REM   display HGR2
440  FOR D = 1 TO 200: NEXT D
450  GOTO 410
```





Line 20 sets up HGR with full screen graphics, while lines 30-110 draw the standing figure. Line 210 moves to HGR2, and the jumping figure is drawn by lines 220-300. Line 410 displays HGR, which is shown for a brief period of time because of the delay loop at line 420. Then line 430 causes HGR2 to be displayed. After the delay loop at line 440 this process is continually repeated to give the figure the appearance of doing jumping jacks.

Exercises

1. Have the computer draw a solid orange rectangle with its upper left corner at (19,10), a length of 3 columns and a height of 2 rows.

2. Have the computer draw a green letter L about two inches high in the upper left corner of the screen.

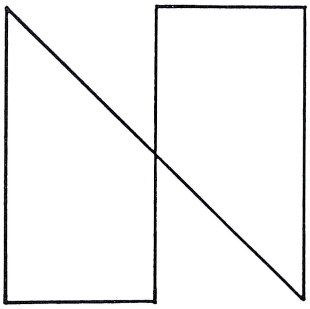
3. Have the computer draw a small letter i about an inch high with the dot at (15,18).

4. Using high resolution mode draw a blue vertical line and a white horizontal line which intersect at (140,80).

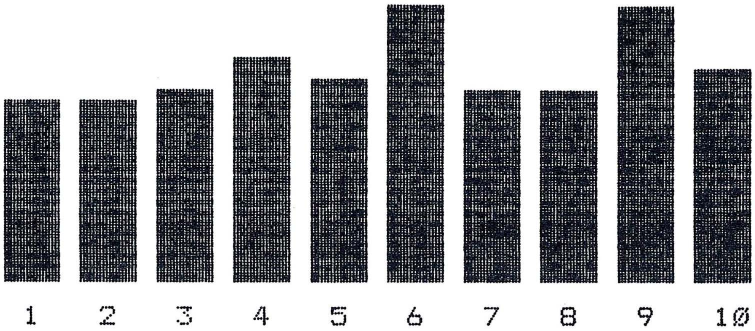
5. Have the computer construct a blue right triangle with the altitude from (250,20) to (250,110) and the base from (250,110) to (130,110). Use only one HPLOT statement.

6. In high resolution mode draw a violet letter A with the top at (40,40) and the lowest point of the left side at (10,130).

7. Using high resolution graphics, have the computer produce the following logo:



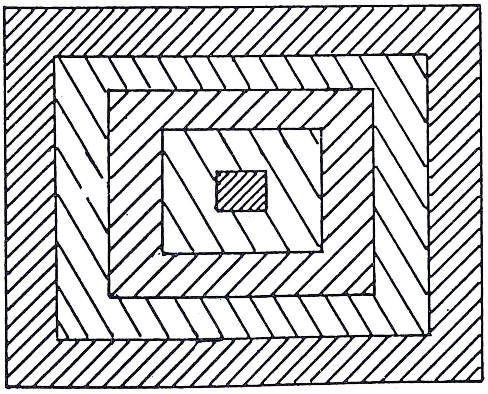
8. Have the computer generate 200 random integers between 1 and 10, inclusive. Using low resolution graphics, plot a properly labeled bar graph showing the number of occurrences of each random number.



9. The following table shows production output per day for each employee of Fazioli's Pizza Parlor. Plot a bar graph showing the output for the week for each of Fazioli's employees:

Employee	Pizza Production
Papa	18, 12, 9, 10, 16, 22, 14
Dellacona	12, 21, 19, 16, 28, 20, 22
Ricardo	18, 20, 14, 19, 11, 16, 23
Ferrara	23, 27, 18, 16, 21, 14, 24

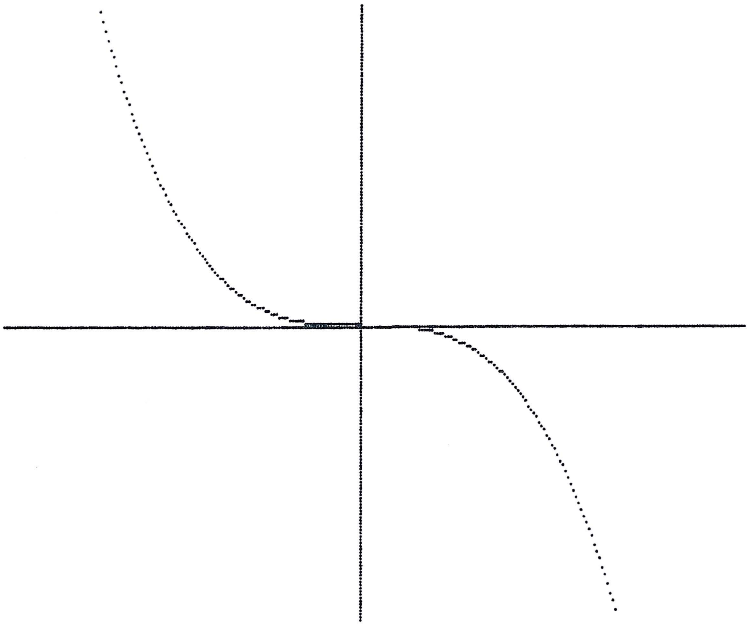
10. Using low resolution graphics, have the computer generate an 18 x 18 rectangular dart board using the colors 1 through 5, with gray (COLOR 5) occupying the center of the board. Each color from 4 to 1 should form a rectanglar border around the colors inside it. Have the computer take 10 random shots at points on the board. Using the SCRN function, total the computer's score (i.e., if the point of impact is within the gray area, the computer scores 5 points; if within the dark green area, the computer scores 4 points, etc.). Show where each dart hits by plotting a black brick at the point of impact.



11. Have the computer fill the first 20 columns of the low resolution screen with randomly colored bricks.

    (a) Using the SCRN function, duplicate the left portion of the screen on the right side of the screen in columns 20 through 39.
    (b) Rewrite part (a) such that the right portion of the screen is a mirror image of the left portion.

12. Using program 6.8 as a guide, plot the graph of Y = −X ∧ 3/10000.

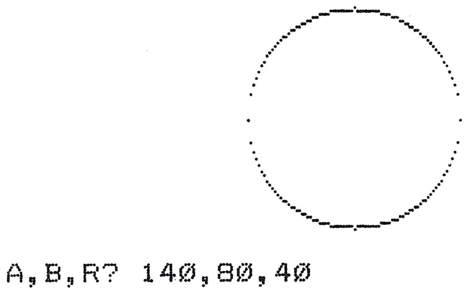


13. Write a program which will simulate the roll of a single die and which will show a picture of the die on the screen. The die should be re-rolled with each press of any key.

14. A programmer can make computer animation by flashing pictures on a screen in quick succession.
    (a) Write a program that will make an arrow similar to the one shown below:

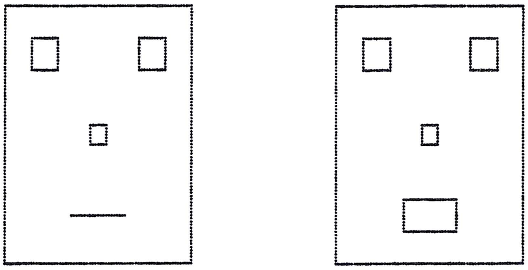
    (b) Make the arrow appear to move across the screen by continuously redrawing it a few spaces to the right each time.

15. Allow the user to enter 3 numbers: A, B, and R. Have the computer plot a circle of radius R with its center at (A,B). (Hint: The equation of a circle with center (A,B) and radius R is $(X-A)^2 + (Y-B)^2 = R^2$.)

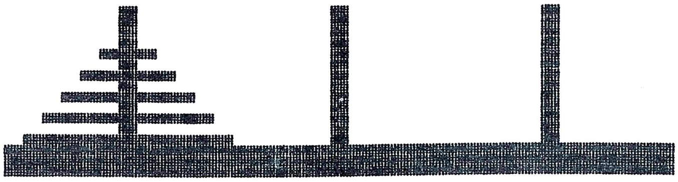


```
A,B,R? 140,80,40
```

16. Draw one of the following figures in HGR and the other in HGR2. Then, using POKE, make the figure to be opening and closing its mouth by alternately displaying each picture.



17. The ancient puzzle "Towers of Hanoi" uses 5 different sized disks and 3 pegs. The 5 disks are initially stacked on the left peg in order of decreasing size. The object of the game is to move all of the disks to the right peg. Only one disk can be moved at a time and a larger disk cannot be placed on top of a smaller disk.

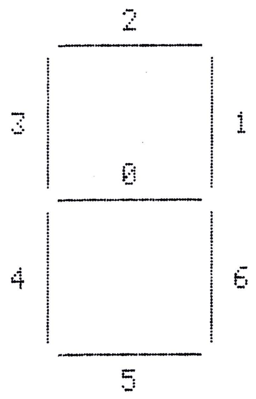


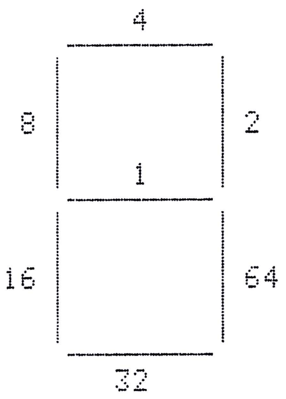
Initial Configuration for "Towers of Hanoi"

(a) Write a program using low resolution graphics that will draw a green base with green pegs and plot the initial configuration of 5 red disks on the left tower.

(b) Expand the program from part (a) to allow movement of the disks. The user should be able to enter two values (P and Q). The should cause the top disk on tower P to be moved to the top of the stack on tower Q. (Hint: Set up a subscripted variable A(X,Y) with maximum. values X=3, Y=5.) There are 15 possible slots for disks; 3 towers, 5 slots per tower. When a slot is occupied, the subscripted variable A(X,Y) should contain the disk number occupying that slot (1-5). If the slot is free, the appropriate position in A(X,Y) should contain a zero).

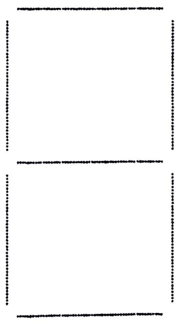
18. A seven segment display is used to represent a number on a digital clock. The seven segments are arranged as shown below in figure-a (numbered zero through six), and when certain segments are lit, they form the figure of a numeral:



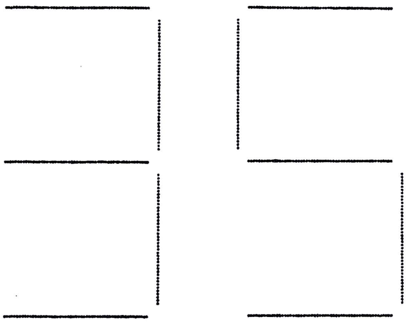


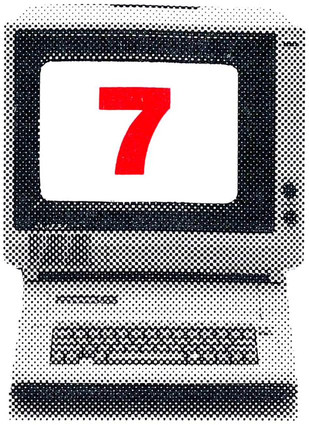
For example, if a figure five is to be represented, then segments 0, 2, 3, 5, and 6 would be lit. If each segment is assigned a value of a power of two as shown in figure-b, it is possible to encode the form of a numeral as the sum of the values of the segments that must be lit to represent that numeral. For example, the encoded information of a figure five would be the sum of segments 0, 2, 3, 5, and 6 or 1 + 4 + 8 + 32 + 64 = 109, to decode a number like 109, successively divide it by two. If the result of the first division is not an integer, then the segment numbered zero should be lit. Then take the INT of the result and again divide by two. If on this second pass the result is again not an integer, then the segment numbered one should be lit. Again take the INT of the result. Continue this process of dividing by two and checking the result seven times (once for each segment).

(a) Write a program to produce one seven segment display as described above, and program the computer to display slowly the sequence from zero to nine.



(b) Modify part (a) of this problem to incorporate a second seven segment display and have the computer count slowly to ninety-nine.





MATHEMATICAL FUNCTIONS

SQR

SGN

ABS

SIN

COS

TAN

ATN

LOG

EXP

DEF

This chapter covers the various mathematical functions that the computer can perform. The extent to which they can be useful to an individual depends on his or her mathematical background. Generally speaking, the functions presented here are primarily employed by mathematicians, scientists and engineers.

SQR

The SQR function calculates the positive square root of a number:

```
10 PRINT SQR(14)
```

The square root of a number N is defined as that value which, when multiplied by itself, gives N. For instance, the square root of 36 is 6 because 6×6 = 36.

Program 7.1

This program illustrates the square root function:

```
10  FOR X = 1 TO 4
20  PRINT "THE SQUARE ROOT OF ";X;" IS "; SQR (X)
30  NEXT X

]RUN
THE SQUARE ROOT OF 1 IS 1
THE SQUARE ROOT OF 2 IS 1.41421356
THE SQUARE ROOT OF 3 IS 1.73205081
THE SQUARE ROOT OF 4 IS 2

]
```

In the expression SQR(X), SQR is the function name and X is called the argument. With the SQR function, the argument may be any mathematical expression with a non-negative value. For example, SQR(3*X + 5) is perfectly acceptable provided that the expression 3*X + 5 produces a non-negative value.

Note that the argument of a function must be enclosed in parentheses. In its evaluation of a mathematical function, the computer first evaluates the argument and then the function. Some functions have a limitation on the value of the argument. Such limitations will be indicated as the functions are introduced.

SGN and ABS

In some situations it may be necessary to know if a variable is positive or negative. The SGN function has only three possible values: 1, 0 and −1.

SGN(X) = 1 if X>0
SGN(X) = 0 if X=0
SGN(X) = −1 if X<0

Program 7.2

This program finds the SGN (signum) of the numbers from 6 to -6, incrementing by -1.2:

```
10  FOR X = 6 TO  - 6 STEP  - 1.2
20 A =  SGN (X)
30  PRINT "SGN(";X;") = ";A
40  NEXT X

]RUN
SGN(6)  = 1
SGN(4.8)  = 1
SGN(3.6)  = 1
SGN(2.4)  = 1
SGN(1.2)  = 1
SGN(0)  = 0
SGN(-1.2)  = -1
SGN(-2.4)  = -1
SGN(-3.6)  = -1
SGN(-4.8)  = -1

]
```

Note that X does not reach -6. This is due to rounding error.

The ABS function can be used to find the absolute value of a number. The ABS function yields values according to the rule:

$$\begin{array}{ll} \text{ABS}(X) = X & \text{if } X>=0 \\ \text{ABS}(X) = -1*X & \text{if } X<0 \end{array}$$

Program 7.3

This program illustrates the use of the ABS function:

```
10  FOR X = 6 TO  - 6 STEP  - 1.2
20 A =  ABS (X)
30  PRINT A;"  ";
40  NEXT X

]RUN
6  4.8  3.6  2.4  1.2  0  1.2  2.4  3.6  4.8
]
```

Again, X does not reach -6 due to rounding error.

Review

1. What is X when

$$X = \text{SQR}\ (\text{ABS}(\text{SGN}(-9+9/3)-6)+9)?$$

2. Write a program to calculate the distance between any 2 points in a Cartesian coordinate system. Hint: use the formula:

$$D = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

```
]RUN
FIRST COORDINATE (X1,Y1)? 1,1
SECOND COORDINATE (X2,Y2)? 4,5
DISTANCE IS 5

]
```

Trigonometric Functions: SIN, COS, TAN

The computer is able to find the values of several trigonometric functions. The functions SIN(X), COS(X) or TAN(X) will produce the value of the sine, cosine or tangent of the angle X, where X is measured in radians. To convert an angle from degrees to radians, multiply it by 3.141592654 and then divide the result by 180 (180 degrees equals π radians).

Program 7.4

The following program illustrates these functions:

```
10 PI = 3.141592654
20  INPUT "VALUE? ";A
30 X = A * PI / 180
40  PRINT A;" DEGREES EQUALS ";X;" RADIANS."
50  PRINT "SIN(";A;") = "; SIN (X)
60  PRINT "COS(";A;") = "; COS (X)
70  PRINT "TAN(";A;") = "; TAN (X)
80  PRINT
90  GOTO 20

]RUN
VALUE? 30
30 DEGREES EQUALS .523598776 RADIANS.
SIN(30) = .5
COS(30) = .866025404
TAN(30) = .577350269

VALUE? 45
45 DEGREES EQUALS .785398163 RADIANS.
SIN(45) = .707106781
COS(45) = .707106781
TAN(45) = 1

VALUE?

BREAK IN 20
]
```

ATN

The only inverse trigonometric function supplied by the computer is the principal arctangent function ATN. The function ATN(X) is used to find the angle whose tangent is X. The value produced by the ATN function is in radians. Thus, the arctangent of 1 is $\pi/4$ radians = .785398163. To convert an angle from radians to degrees, multiply it by 180 and then divide the result by π. The ATN function, just like the principal arctangent function in mathematics, gives values only between $-\pi/2$ and $\pi/2$ radians. There is no limitation on the value that the argument may assume.

Program 7.5

This program finds the angle whose tangent is entered and prints the result in degrees:

```
10 PI = 3.141592654
20  INPUT "ENTER A TANGENT VALUE. ";T
30 R =  ATN (T)
40 D = R * 180 / PI
50  PRINT "THE ANGLE WHOSE TANGENT IS ";T;" IS ";D;" DEGREES."
60  PRINT : GOTO 20

]RUN
ENTER A TANGENT VALUE. 1
THE ANGLE WHOSE TANGENT IS 1 IS 45 DEGREES.

ENTER A TANGENT VALUE. 0
THE ANGLE WHOSE TANGENT IS 0 IS 0 DEGREES.

ENTER A TANGENT VALUE. .57735
THE ANGLE WHOSE TANGENT IS .57735 IS 29.9999884 DEGREES.

ENTER A TANGENT VALUE.

BREAK IN 20
]
```

To find the principal arcsine of a number, the programmer must use the following identity:

$$\text{ARCSINE (X)} = \text{ARCTAN}\left(\frac{X}{\sqrt{1 - X^2}}\right)$$

Therefore, to find the principal angle whose sine is X, the expression ATN(X/SQR(1 − X↑2)) is used. This angle will be measured in radians and will be between $-\pi/2$ and $\pi/2$. The value of X, however, must be between −1 and 1, not inclusive.

To find the arc cosine of X, use the expression ATN(SQR(1−X↑2)/X). This gives the angle, which is between $-\pi/2$ and $\pi/2$ whose cosine is X. In this expression, X must be between −1 and 1, inclusive, not equal to 0.

Program 7.6

This program finds the arcsine and arccosine of X in both radians and degrees:

```
10 PI = 3.141592654
20  INPUT "VALUE? ";X
30 S =  ATN (X /  SQR (1 - X ^ 2))
40 S1 = S * 180 / PI
50 C =  ATN ( SQR (1 - X ^ 2) / X)
60 C1 = C * 180 / PI
70  PRINT "THE ANGLE WHOSE SIN IS ";X;" IS ";S;" RADIANS."
80  PRINT "THE ANGLE WHOSE COSINE IS ";X;" IS ";C;" RADIANS."
90  PRINT "THE ANGLE WHOSE SIN IS ";X;" IS ";S1;" DEGREES."
100  PRINT "THE ANGLE WHOSE COSINE IS ";X;" IS ";C1;" DEGREES."
110  PRINT : GOTO 20

]RUN
VALUE? .5
THE ANGLE WHOSE SIN IS .5 IS .523598776 RADIANS.
THE ANGLE WHOSE COSINE IS .5 IS 1.04719755 RADIANS.
THE ANGLE WHOSE SIN IS .5 IS 30 DEGREES.
THE ANGLE WHOSE COSINE IS .5 IS 60 DEGREES.

VALUE? .877
THE ANGLE WHOSE SIN IS .877 IS 1.06958255 RADIANS.
THE ANGLE WHOSE COSINE IS .877 IS .501213777 RADIANS.
THE ANGLE WHOSE SIN IS .877 IS 61.2825659 DEGREES.
THE ANGLE WHOSE COSINE IS .877 IS 28.7174341 DEGREES.

VALUE?

BREAK IN 20
]
```

Logarithms and the Exponential Function: LOG, EXP

The LOG function can be used to find natural logarithms, that is, logarithms to the base e. To find the natural logarithm of X, LOG(X) is used. Do not confuse the natural logarithm with the common logarithm usually studied in a second year algebra course. The common logarithm, that is, logarithm to base 10, can be found from the natural logarithm by using the formula:

$$\log_{10}(X) = \frac{\ln(X)}{\ln(10)}$$

In the above ln(X) designates the natural logarithm of X. Therefore, to find $\log_{10}(X)$, simply use LOG(X)/LOG(10). The argument in the LOG function must always be positive.

The function EXP(X) is used to find values of the exponential function e^x, where e = 2.71828. This number is the same as the base of the natural logarithm function.

Program 7.7

This program illustrates the use of the above functions:

```
10  INPUT "ENTER X. ";X
20  PRINT "LN(X) = "; LOG (X)
30  IF X <  = 88 THEN  PRINT "e RAISED TO X = "; EXP (X)
40 T =  LOG (X) /  LOG (10)
50  PRINT "THE COMMON LOGARITHM OF ";X;" IS ";T
60  PRINT : GOTO 10

]RUN
ENTER X. 1
LN(X) = 0
e RAISED TO X = 2.71828183
THE COMMON LOGARITHM OF 1 IS 0

ENTER X. .01
LN(X) = -4.60517019
e RAISED TO X = 1.01005017
THE COMMON LOGARITHM OF .01 IS -2

ENTER X. 10000
LN(X) = 9.21034038
THE COMMON LOGARITHM OF 10000 IS 4

ENTER X. .525
LN(X) = -.644357016
e RAISED TO X = 1.69045885
THE COMMON LOGARITHM OF .525 IS -.279840697

ENTER X.

BREAK IN 10
]
```

Line 30 is necessary to avoid the overflow error that occurs when X is larger than 88.

DEF

Several standard functions, such as SQR, ABS and LOG, have already been introduced in this chapter. In addition, the programmer can define other functions by using a DEF statement. The major advantage of DEF lies in the fact that the expression for the function need only be written once, even though the function can be evaluated at more than one location within the program. The form of a DEF statement is:

DEF FN <function name> (<variable name>) = <expression>

The function name may be any acceptable numeric variable name (e.g., FNA, FNF, FNG3). The variable name (i.e., the argument) following the function name must always appear within parentheses and may be any appropriate numeric variable. In the following example,

10 DEF FNP(X) = X↑2 − 2∗X − 1

P is the function name, X is the variable name, and $X^2 - 2X - 1$ is the expression used to compute the function's value. For instance, when X = 5, FNP(X) = 14 because 5↑2 − 2*5 − 1 = 14.

Program 7.8

The following program evaluates the polynomial function FNP(X) several times:

```
10  DEF  FN P(X) = X ^ 2 - 2 * X - 1
20  PRINT "X","FNP(X)"
30  FOR A = 1 TO 5
40  PRINT A, FN P(A)
50  NEXT A
60  INPUT X
70  PRINT "THE RESULT OF FNP(";X;") IS "; FN P(X)

]RUN
X               FNP(X)
1               -2
2               -1
3               2
4               7
5               14
?-10
THE RESULT OF FNP(-10) IS 119

]
```

When the function is evaluated at line 40, the variable in parentheses is A. When it is evaluated at line 70, the variable is X. The name of the variable in parentheses may be the same as or different from the variable name used in the DEF statement. Note also that if the DEF statement were not used, the formula on line 10 would have to appear twice (lines 40 and 70). Though the function is defined only once, it may be evaluated anywhere within the program, providing program economy.

Another advantage of the DEF statement is that it can be easily retyped to define a different function. This is illustrated by re-running Program 7.8 with line 10 changed to:

```
]10 DEF FN P(X) = X ^ 3 - 5 * X ^ 2 + 1

]RUN
X               FNP(X)
1               -3
2               -11
3               -17
4               -15
5               1
?-10
THE RESULT OF FNP(-10) IS -1499

]
```

Review

3. Write a program that will produce the following output. Two user defined functions should be used: one to convert degrees to radians, the other to convert radians to degrees:

```
]RUN
DEGREES? 30
THAT IS .523598775 RADIANS.

RADIANS? .785375
THAT IS 44.9986729 DEGREES.

]
```

Shifting and Scaling a Graph

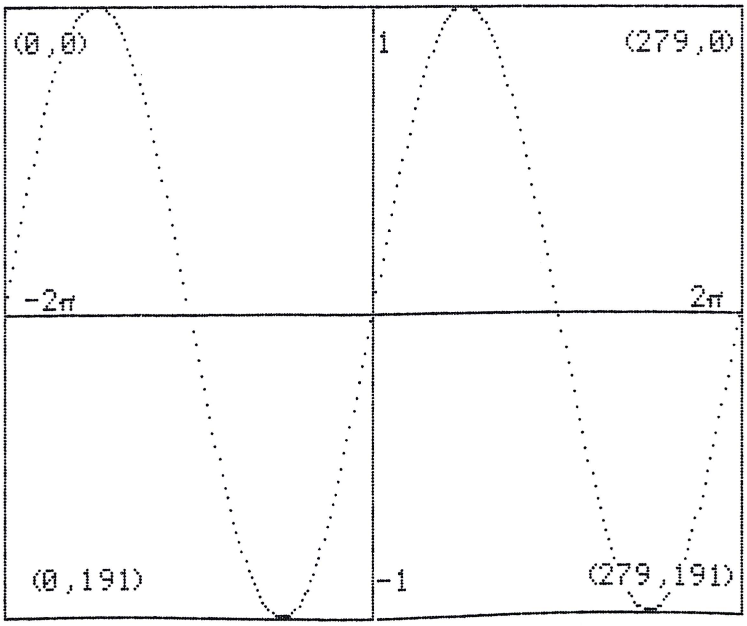
A hand-drawn graph usually includes an X-axis drawn horizontally, a Y-axis drawn vertically, and a continuous curve drawn through a number of plotted points. There are some obvious limitations to graphs produced by the computer, (for example, the lack of connection between plotted points). The graphing area on the display screen contains a limited number of points available for plotting. Thus, certain adjustments of the X and Y-axes may be necessary to accommodate functions which produce values that exceed the number of graphing points available. This problem is solved by determining the domain (smallest and largest X values) and the range (smallest and largest Y values) for a function where Y = f(X) and then adjusting the axes accordingly. These processes are referred to as shifting and scaling.

Shifting is the process of moving a graph into an acceptable plotting range. This is done by setting the minimum value of the domain to zero by adding some constant, and setting the maximum value of the range to zero by subtracting from some constant.

Scaling is the process of limiting the domain and range to the boundaries of the computer screen. This is done by setting the maximum value of the domain to 279, and the minimum value of the range to 191.

This process of shifting and scaling is demonstrated by the following example:

To plot the function Y = SIN(X) for a domain of -2π to 2π and a range of -1 to 1 the coordinates must be shifted and scaled to the computer's X coordinates of 0 to 279 and Y coordinates of 191 to 0.



To shift the domain of this function add 2π to all the X values. This gives a new domain of 0 to 4π. To scale this new domain multiply all these values by $(279/4\pi)$ so that the plotting domain is from 0 to 279.

To shift the range of this function the Y values must be subtracted from 1 to give a new range of 2-0. To scale this new range multiply by (191/2) to obtain the plotting range of 191-0.

In general, this process is described by the following formulas:

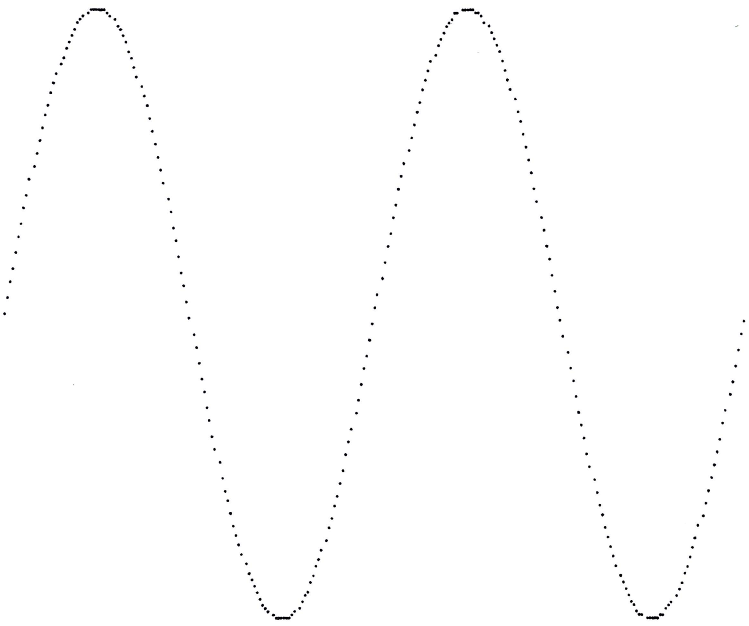
$$X0 = (X - \mathrm{Xmin}) * 279/(\mathrm{Xmax} - \mathrm{Xmin})$$
$$Y0 = (\mathrm{Ymax} - Y) * 191/(\mathrm{Ymax} - \mathrm{Ymin})$$

(X,Y) is the true function coordinate, and (X0,Y0) is the corresponding coordinate for the point to be plotted on the computer screen. Xmin to X max is the function's domain and Ymin to Ymax is the function's range.

Program 7.9

The following program illustrates how the function Y = 150*SIN(X) is shifted and scaled to fit on the full high resolution screen.

```
10  HGR2 : HCOLOR= 3:PI = 3.141592654
20  FOR X =  - 2 * PI TO 2 * PI STEP .05
30 Y = 150 *  SIN (X): REM  the function
40  REM
50  REM   shift and scale the X-axis
60 X0 = (X + 2 * PI) * (279 / (4 * PI))
70  REM
80  REM  shift and scale the Y-axis
90 Y0 = (150 - Y) * (191 / 300)
100  REM
110  HPLOT X0,Y0
120  NEXT X

]RUN
```

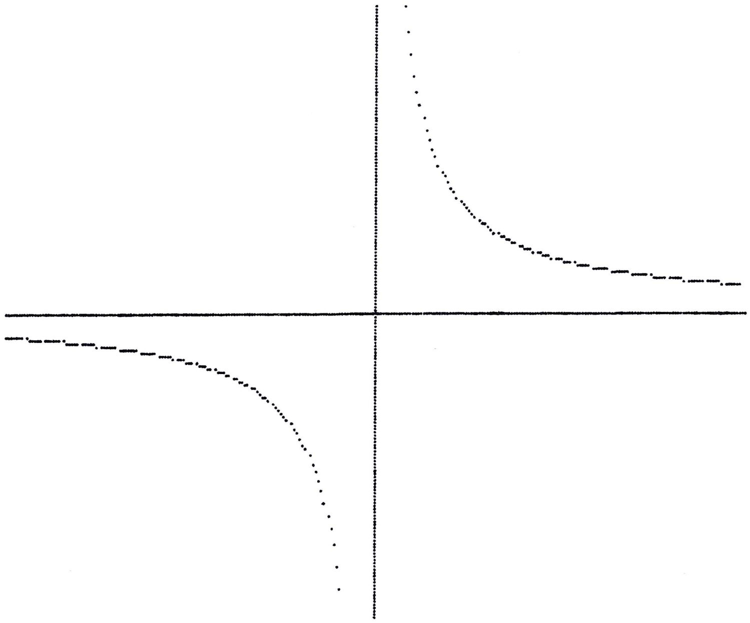


The function is given a domain of -2π to 2π in increments of 0.05 by line 20. The result (y) is evaluated at line 30 and has a range of -150 to 150. The X value is shifted and scaled by line 60 and the Y value by line 90.

Program 7.10

This program plots the function Y = 1/X on the display screen. Note how the graph has been shifted and scaled.

```
10  HGR2 : HCOLOR= 3
15  REM  plot axes
20  HPLOT 140,0 TO 140,191
30  HPLOT 0,96 TO 279,96
35  REM  plot function
40  FOR X =  - 12 TO 12 STEP .1
50  IF X = 0 THEN 120: REM  invalid argument to function
60 Y = 1 / X: REM  function
70 X0 = (X + 12) * (279 / 24): REM  shift and scale X-axis
80 Y0 = (1 - Y) * (191 / 2): REM  shift and scale Y-axis
90  IF X0 < 0 OR X0 > 279 THEN 120: REM  unplottable points
100  IF Y0 < 0 OR Y0 > 191 THEN 120: REM  unplottable points
110  HPLOT X0,Y0
120  NEXT X

]RUN
```



The analysis of this program is left as an exercise for the reader.

Exercises

1. When a number is input, have the computer generate the following output. Be sure to account for a negative input:

```
]RUN
?-1
No negative numbers allowed.
?4
N=4             Square roots = + or -2
?
```

2. Write a program which prints the integers from 121 to 144, inclusive, and their respective square roots. Label each column of the output.

3. Write a program that will print COS(X) and SIN(X+90°), where X varies from 0 to 360 degrees in increments of 10 degrees.

4. Input a number N. If N is zero, print 0. Otherwise, print ABS(N)/N. What does the program do?

5. Input a number N and print the product of SGN(N) and N. What does this program do?

6. Input a number N, square it and print the square root of the result. What should the program produce?

7. Print a table consisting of 2 columns with headings showing each angle in radians and degrees. The angles in radians are to be 0, .25, .5, .75, . . ., 3.0. Remember that 180° = π radians.

8. Input an angle in degrees and convert it to a fraction of a revolution (1 rev. = 360°) and to radians.

9. Input an angle in degrees. Of the three functions sine, cosine, and tangent, print the value of the one which has the greatest value.

10. For angles from 0° to 180° (at intervals of 10°) print the angle in degrees, the sine, the cosine, and the sum of their squares in columns with headings. What patterns emerge?

11. Input two numbers (A,B). Print the quantity F(B)−F(A), given that $F(X) = 9X^3 - 7X^2 + 4X - 1$.

12. Input a number N. Print the values of F(N) and F(F(N)), where F(X) = 20 * SQR (ABS(X)) − 10 * SGN(X) + 5 * INT(X).

13. Print a table (with headings) of X, the natural logarithm of X and the exponential function of X for X = 1 to 15.

14. Print a table (with headings) of X, the logarithm of X to the base 10, and 10 raised to the power X for X = 1 to 15.

15. What is the exact output for the following program? Check by running the program:

```
10  READ A,B,C,D
20  PRINT  SQR (A), INT (B), SQR ( INT (C)), INT ( SQR (D))
30  DATA  25,-3.4,9.7,24
```

16. Print the square root of the integers from 50 to 60, inclusive.

17. What is the exact output for the following program? Check by running the program:

```
10  DEF  FN F(N) = 3 * N - 6
20  FOR X =  - 4 TO 6 STEP 2
30  IF  FN F(X) > 0 THEN  PRINT "
     FNF(";X;") is positive"
40  IF  FN F(X) = 0 THEN  PRINT "
     FNF(";X;") is zero"
50  IF  FN F(X) < 0 THEN  PRINT "
     FNF(";X;") is negative"
60  NEXT X
```

18. Input a number and print the square root, sign, log and sine of the number. For example:

```
]RUN
?16
SQR(16) = 4
SGN(16) = 1
LOG(16) = 2.77258872
SIN(16) = -.287903315

]
```

19. What is the exact output for the following program? Check by running the program:

```
10  FOR X =  - 3 TO 4
20  READ A
30  PRINT  SGN (X) *  ABS (X)
40  NEXT X
50  DATA  3,-5,1,6,-2,4,-9,5
```

20. Using three user-defined functions, have the computer evaluate the following for the integers from −10 to 10:

$$X^2 + 3X + 2$$
$$LOG(X^2 + 1) - X$$
$$ATN(SIN(X))$$

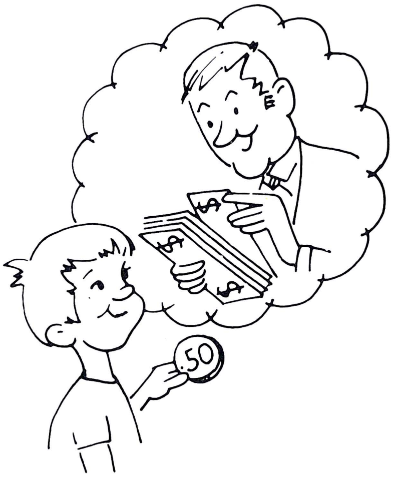
21. Write a program to convert from polar to rectangular coordinates (i.e., from (r,Θ) to (X,Y)).

22. If two functions, f and g, are inverse to each other, the following relations hold: f(g(x)) = x and g(f(x)) = x.

(a) Tabulate the values of the following quantities for X = −5 to 10: X, EXP(X), LOG(EXP(X)).
(b) Print a table for X = 1 to 151 of X, LOG(X), EXP(LOG(X)), using STEP 10.
(c) Do EXP and LOG appear to be inverse to each other?

23. Produce your own sequence of random numbers without using the RND function. To do this, let X vary from 1 to 100 in steps of 1. Obtain SIN(X) and multiply this by 1000, calling the absolute value of the product Y. Divide INT(Y) by 16, and let the remainder R serve as your random number. Hint: the remainder of A ÷ B is A/B − INT(A/B).

24. Six year old Dennis the Menace has decided to invest 50¢ in the Last Chew Bubble Gum Company. Starting with the 11th year, he withdraws 5¢ at the beginning of each year. His money earns 8% interest compounded continually. The formula for interest compounded continually is $P = P_0e^{it}$, where t is the elapsed time in years, P_0 is the initial deposit, P is the amount at time t, and i is the interest rate. In this case the formula would be $P = P_0e^{.08t}$. How much is Dennis's deposit worth after 50 years?



25. Use the SIN function to generate the following:

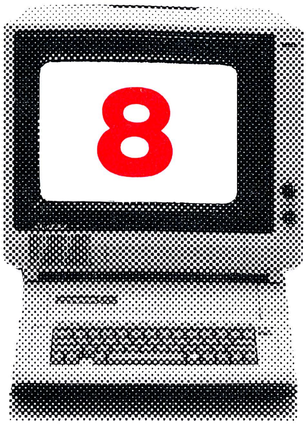
```
RUN
                 SHAZAM!
                    SHAZAM!
                       SHAZAM!
                          SHAZAM!
                            SHAZAM!
                              SHAZAM!
                               SHAZAM!
                                SHAZAM!
                                SHAZAM!
                                SHAZAM!
                               SHAZAM!
                             SHAZAM!
                           SHAZAM!
                         SHAZAM!
                      SHAZAM!
                   SHAZAM!
                SHAZAM!
            SHAZAM!
         SHAZAM!
       SHAZAM!
    SHAZAM!
   SHAZAM!
 SHAZAM!
 SHAZAM!
 SHAZAM!
 SHAZAM!
  SHAZAM!
    SHAZAM!
      SHAZAM!
         SHAZAM!
            SHAZAM!
               SHAZAM!

READY.
```

26. Write a program that will solve a triangle (computer the unknown sides and angles) for the following situations:
    (a) given two sides and the included angle,
    (b) given two angles and any side,
    (c) given three sides.

27. Print a graph of $Y = \text{SIN}(n\pi X)$, where n is chosen by the user from the set (−1, 1, ½ ¼). The units of length for the X and Y axes need not be the same. The periodic nature of the sine function should be indicated on your graph.

28. Produce the following graphs with X and Y axes and the origin properly centered.

    (A) Print a graph of Y = SGN(X)
    (B) Print a graph of Y = ATN(X)
    (C) Print a graph of Y = INT(X)



STRING FUNCTIONS AND DATA TYPES

ASC

CHR$

LEFT$

LEN

MID$

RIGHT$

STR$

VAL

Modern computers have resulted from the union of the binary system with the principles of electricity. The binary number system uses only two digits, 0 and 1, and can therefore be represented by the two states of an electric circuit, off or on. This concept has made digital computers possible because they operate by reducing all data to a binary code. How different types of data can be reduced to a binary system is explained in this chapter.

Binary Code

The most familiar number system, the decimal system, uses ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and is therefore considered to be base ten. In contrast, the binary system, which uses only the digits 0 and 1, is base two.

In the base ten system, columns are used to represent powers of ten with the first column left of the decimal point representing 10^0, the second 10^1, the third 10^2 and so on. For example, in the numeral 458, 8 represents $8x10^0$, 5 represents $5x10^1$ and 4 represents $4x10^2$. The numeral itself represents the sum: $4x10^2 + 5x10^1 + 8x10^0$.

The base two system works identically *except the columns represent powers of two* instead of ten. For example, in the numeral 101, the 1 on the right represents $1x2^0$, the 0 represents $0x2^1$ and the 1 on the left represents $1x2^2$. The numeral itself represents the sum $1x2^2 + 0x2^1 + 1x2^0$, which is known as five in the base ten system.

Base Two	**Base Ten**
$11 = 1x2^1 + 1x2^0$	$= 3$
$1011 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$	$= 11$
$11001 = 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$	$= 25$

To convert a number from base ten to base two, we must find which powers of two add up to the number. Since $13 = 8 + 4 + 1$, the base two representation for 13 is 1101 ($1x8 + 1x4 + 0x2 + 1x1$).

Base Ten	**Base Two**
$6 = 4 + 2 = 1x2^2 + 1x2^1 + 0x2^0$	$= 110$
$29 = 16 + 8 + 4 + 1 = 1x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$	$= 11101$
$52 = 32 + 16 + 4 = 1x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$	$= 110100$

Computer Memory and Processing

The computer is composed of a solid-state electronic memory, which stores information, and a central processing unit (CPU), which performs calculations, makes decisions and moves information.

Because electricity has two states, ON and OFF, it is ideal for expressing binary numbers. A circuit (called a "flip-flop") when on stands for a "1" and when off stands for a "0". Designing computers to contain millions of simple flip-flop circuits allows huge quantities of information to be stored.

A single binary digit (0 or 1) is called a "bit", and eight of these bits constitute a "byte". Single characters require one byte and integers two bytes of memory for storage. The memory stores both instructions for the CPU and data as binary digits.

The power of a computer is vastly increased when it is capable of storing letters and special characters as well as numbers. In order to do this, a code has been established to translate letters and special characters into numbers, which can then be stored in a binary form. This code has been standardized by the computer industry as the American Standard Code for Information Interchange (ASCII). In this code each letter of the alphabet, both upper case and lower case, each symbol, and each control function used by the computer is represented by a number. For example, the name JIM is translated by the computer into ASCII numbers 74,73,77. In turn these numbers are then stored by the computer in binary form:

J = 74 = 01001010
I = 73 = 01001001
M = 77 = 01001101

In order for the computer to store a name such as JIM, or any piece of non-numeric information, it must be entered in the form of a string, converted character by character to ASCII numbers, and then stored in memory as binary numbers. On the following page is a table of the ASCII character codes available on the computer:

01001010
01001001

Decimal Value	ASCII Character	Usage	Decimal Value	ASCII Character	Usage	Decimal Value	ASCII Character	Usage
0	NUL	FILL character	43	+		86	V	
1	SOH		44	,	COMMA	87	W	
2	STX		45	-		88	X	
3	ETX	CTRL/C	46	.		89	Y	
4	EOT		47	/		90	Z	
5	ENQ		48	0		91	[	
6	ACK		49	1		92	\	
7	BEL	BELL	50	2		93	]	
8	BS		51	3		94	ˆ or ↑	
9	HT	HORIZ. TAB	52	4		95	_ or ←	
10	LF	LINE FEED	53	5		96	`	Grave accent
11	VT		54	6		97	a	
12	FF		55	7		98	b	
13	CR	CAR. RET	56	8		99	c	
14	SO		57	9		100	d	
15	SI	CTRL/O	58	:		101	e	
16	DLE		59	;		102	f	
17	DC1	CTRL/Q	60	<		103	g	
18	DC2		61	=		104	h	
19	DC3	CTRL/S	62	>		105	i	
20	DC4		63	?		106	j	
21	NAK		64	@		107	k	
22	SYN		65	A		108	l	
23	ETB		66	B		109	m	
24	CAN		67	C		110	n	
25	EM		68	D		111	o	
26	SUB		69	E		112	p	
27	ESC	ESCAPE	70	F		113	q	
28	FS		71	G		114	r	
29	GS		72	H		115	s	
30	RS		73	I		116	t	
31	US		74	J		117	u	
32	SP	SPACE	75	K		118	v	
33	!		76	L		119	w	
34	″		77	M		120	x	
35	#		78	N		121	y	
36	$		79	O		122	z	
37	%		80	P		123	{	
38	&		81	Q		124	|	Vertical Line
39	‘	APOSTROPHE	82	R		125	}	
40	(		83	S		126	˜	Tilde
41	)		84	T		127	DEL	DELETE
42	*		85	U				

Data Types

When information is entered into the computer, it can take one of three forms: ASCII characters, integers or floating point numbers. An integer is a number without decimal places, while a floating point number has either a decimal point and decimal places or an implied decimal point. For example, the number 29 could be either integer or floating point, but 29.73 is definitely floating point. Floating point numbers are internally accurate to ten significant figures but rounded to nine when printed.

Storage of an integer requires 16 bits, a floating point number 32 bits, and an ASCII character 8 bits. Because of storage requirements it is important to distinguish among the several forms. When files are presented in Chapter Nine, the significance of this factor in planning efficient storage of data will become apparent.

The three data types employ different symbols to inform the computer as to which is being used. Strings of ASCII characters employ the familiar symbol ($) for string variable names (e.g., G$, B3$). Integers, which have the disadvantage of being restricted to the range −32767 to 32767, are denoted by a percent sign (%). B% and Z% are variable names for integers. While integers have a severe range limitation, they are processed fastest by the computer. Variables and constants not specifically declared otherwise are assumed to be floating point (e.g., R, L).

Program 8.1

This program demonstrates the difference between floating point and integer arithmetic:

```
10 A% = 4 / 3
20  PRINT A%
30 A = 4 / 3
40  PRINT A

]RUN
1
1.33333333

]
```

The computer approximates the fraction 4/3 as 1.33333333 and assigns this value to the floating point variable A at line 30. However, as A% can only have integer values, the computer truncates all the numbers after the decimal point and assigns it the value of 1.

Review

1. What are the minimum number of bits required to store the following data in the computer?

(a) 6167
(b) APRIL 1, 1963
(c) 42864
(d) 35.78

ASCII Code-ASCII Character Conversions

Earlier in this chapter the ASCII code was described as a method by which the computer converts information in the form of strings into numbers. Commands exist to convert strings into ASCII values and vice versa:

Function Format	**Operation**
X = ASC(P$)	Converts only the first character of the string P$ to its ASCII code number.
P$ = CHR$(X)	Assigns the character with the ASCII code number of X to P$, which now contains only one character.

Program 8.2

This program demonstrates the use of the ASCII character conversions:

```
1   REM  C$ = character input
2   REM  A = ASCII value of C$
3   REM  K$ = new string
4   REM  Y$ = indicator
5   REM
6   REM
10 K$ = ""
20  INPUT "ENTER ANY CHARACTER ";C$
30 A =  ASC (C$)
40  PRINT "THE ASCII VALUE OF'";C$;"'IS: ";A
50  PRINT "'"; CHR$ (A);"'" REPEATED 17 ";
60  PRINT "TIMES LOOKS LIKE THIS:"
70  FOR X = 1 TO 17
80 K$ = K$ +  CHR$ (A)
90  NEXT X
100  PRINT K$
110  PRINT "WOULD YOU LIKE TO RUN THIS AGAIN";
120  INPUT Y$
130  IF  ASC (Y$) = 89 THEN 10

]RUN
ENTER ANY CHARACTER R
THE ASCII VALUE OF'R'IS: 82
'R' REPEATED 17 TIMES LOOKS LIKE THIS:
RRRRRRRRRRRRRRRRR
WOULD YOU LIKE TO RUN THIS AGAIN?N

]
```

2. Write a program that will allow the user to input a letter. Have the computer print that letter's ASCII code and then print the letter that comes two letters after it in the alphabet. If Y or Z is the letter input, then A or B should be returned respectively.

```
]RUN
INPUT A LETTER? R
THE ASCII OF 'R' IS 82
TWO LETTERS AFTER 'R' IS 'T'

]
```

String Manipulation Functions

Strings can be manipulated with the following functions:

Function	**Operation**
L$ = LEFT$(A$,N)	Assigns L$ a substring of the string A$ from the leftmost character up to and including the Nth character.
L = LEN(A$)	Assigns L a value equal to the number of characters in the string A$, including blank spaces.
M$ = MID$(A$,N)	Assigns M$ a substring of the string A$ from the Nth character to the rightmost character of the string.
M$ = MID$(A$,N1,N2)	Assigns M$ a substring of the string A$, starting with the character N1 and being N2 characters long.
R$ = RIGHT$(A$,N)	Assigns R$ a substring of the string A$ consisting of the rightmost N characters.
S$ = STR$(X)	Converts X into a string of numeric characters and assigns it to S$.
V = VAL(A$)	Converts the first set of numeric characters in the string A$ into a numeric value that is assigned to V. Non-numeric characters are ignored.

Program 8.3

This program will input a string and manipulate it to show the use of string functions:

```
10   INPUT T$
20   PRINT
30   PRINT "'";T$;"' BACKWARDS IS: ";
40   FOR X =  LEN (T$) TO 1 STEP  - 1
50   PRINT  MID$ (T$,X,1);
60   NEXT X
70   PRINT
80   PRINT "THE VALUE OF '";T$;"' IS: "; VAL (T$)
90   PRINT "'";T$;"' HALVED AND FLIPPED IS:"
100  PRINT  RIGHT$ (T$, LEN (T$) / 2); LEFT$ (T$, LEN (T$)
      / 2)
110  PRINT "'"; VAL (T$);"' BACKWARDS IS: ";
120  FOR X =  LEN ( STR$ ( VAL (T$))) TO 1 STEP  - 1
130  PRINT  MID$ ( STR$ ( VAL (T$)),X,1);
140  NEXT X

]RUN
?145RTLMJDQ

'145RTLMJDQ' BACKWARDS IS: QDJMLTR541
THE VALUE OF '145RTLMJDQ' IS: 145
'145RTLMJDQ' HALVED AND FLIPPED IS:
LMJDQ145RT
'145' BACKWARDS IS: 541
]
```

Program 8.4

This program plays the game Hangman with the user. The computer picks a random word from the DATA statements and allows the user to guess letters which may be contained in the word. If the letter guessed is in the word, the computer shows the user the position in which it appears. If the letter guessed is not in the word, it is counted as a mistake. Play continues until the user guesses all the letters in the word or makes 6 mistakes.

```
10   FOR I = 0 TO  INT (10 *  RND (1))
20   READ W$
30   NEXT I
40   PRINT "O.K. I'M THINKING OF A WORD."
50   PRINT :C = 1
60   FOR I = 1 TO  LEN (W$)
70 D$ = D$ + "?"
80   NEXT I
90   PRINT D$
100  PRINT "YOU HAVE ";M;" MISTAKES."
110  INPUT "LETTER ";L$
120  REM
```

```
200  REM          current word status
210  FOR I = 1 TO  LEN (W$)
220  IF  MID$ (W$,I,1) <  > L$ THEN 270
230 C = 0: REM  a correct guess
240  IF I = 1 THEN D$ = L$ +  RIGHT$ (D$, LEN (D$) - 1): GOTO
     270
250  IF I =  LEN (W$) THEN D$ =  LEFT$ (D$, LEN (D$) - 1)
      + L$: GOTO 270
260 D$ =  LEFT$ (D$,I - 1) + L$ +  RIGHT$ (D$, LEN (D$) -
     I)
270  NEXT I
280  IF D$ = W$ THEN 400
290 M = M + C:C = 1
300  IF M = 6 THEN 500
310  GOTO 90
320  REM
400  PRINT "YOU GOT IT!!": END
410  REM
500  PRINT "YOU HAVE TOO MANY MISTAKES."
510  PRINT "THE WORD WAS ";W$;"."
520  REM
600  REM  data section
610  DATA  APPLE,COMPUTER,KEYBOARD,MONITOR
620  DATA   HUTT,JEDI,EWOK,CARBONITE,SABER,FALCON

]RUN
O.K. I'M THINKING OF A WORD.

????
YOU HAVE 0 MISTAKES.
LETTER H
H???
YOU HAVE 0 MISTAKES.
LETTER O
H???
YOU HAVE 1 MISTAKES.
LETTER T
H?TT
YOU HAVE 1 MISTAKES.
LETTER U
YOU GOT IT!!

]
```

The loop from line 10 to 30 chooses the computer's random word (W$) from the list of ten words in the DATA statements. The loop from line 60 to 80 creates a dummy string (D$) which initially contains a question mark for each letter in the computer's secret word. The loop from line 210 to 270 determines whether a letter was guessed correctly. Note that each question mark in D$ is replaced with the appropriate letter each time a correct guess is made.

Exercises

1. Enter a string A$ and use a loop to print the ASCII number of each of its characters.

2. Input three letters, add their ASCII numbers, find the INT of one third of their sum and print the character corresponding to the result. Is there any meaning to this process?

3. Using properly selected ASCII numbers on a DATA line, print the sentence "ASCII DID THIS!"

4. Using ASCII numbers 45 and 46 and the PRINT TAB statement, print the following figure:

```
]RUN
.--------.
.        .
.        .
.        .
.--------.

]
```

5. Enter the string THREE!@#$%STRING!@#$%FUNCTIONS. Use LEFT$, MID$, and RIGHT$ to print the phrase "THREE STRING FUNCTIONS".

6. Enter a string A$ of any length. Print the length of A$ and the ASCII number of its first and last characters.

7. Write a program where you input a name and then have the computer print the ASCII number of the letter which appears most often in the name.

8. Let A$ = "2598." Have the computer apply VAL to those parts of A$ whose sum is 123.

9. Enter the words "QUEEN," "LENGTH" and "REMEMBER." Print each word with all E's removed.

10. Use the computer or a hand calculator to find the binary equivalents of the decimal values given below:

    89, 74, 80, 107, 255, 129, 28, 39, 29, 24, 43

11. Use the computer or a hand calculator to find the decimal equivalents for the binary values given below:

    1011, 10100, 1111, 1110, 1010011, 110011, 1011100,
    1101111, 11000000, 10000111

12. Input a string A$ which consists of the digits 0 to 9, inclusive, each to be used only once (e.g., 1956472038). Use string functions to obtain from A$ two numbers N1 and N2, N1 being the number represented by the first three digits of A$ and N2 the number represented by the last three digits. Print the sum of N1 and N2.

13. Let A$ consist of the first twelve letters of the alphabet. Using A$, construct the right triangle shown at the right:

```
]RUN
A
AB
ABC
ABCD
ABCDE
ABCDEF
ABCDEFG
ABCDEFGH
ABCDEFGHI
ABCDEFGHIJ
ABCDEFGHIJK
ABCDEFGHIJKL

]
```

14. As a young boy Franklin Roosevelt signed his letters to his mother backwards: TLEVESOOR NILKNARF. Write a program that accepts a person's name and prints it backwards.

15. Using the ASCII chart given in the text, determine the output for the following program. Check by running the program:

```
10   FOR A = 1 TO 4
20   READ A$
30   PRINT  ASC (A$)
40   NEXT A
50   DATA  "W","H","7","625"
60   END
```

16. The output of the following program is Foxy Moxie's message to Jimmy Band. What is the message? Check by running the program:

```
10   FOR W = 1 TO 6
20   READ A$,B$,C$
30   PRINT  LEFT$ (A$,1); MID$ (B$,2,2); RIGHT$ (C$,1)
40   NEXT W
50   END
```

```
60  REM    Data section
70  DATA   FDO,NOXP,LUY
80  DATA   SSS,HAYE,RRS
90  DATA   HYF,OELR,EVP
100  DATA   WER,LILO,VEL
110  DATA   CIS,TOMH,EIE
120  DATA   SRS,WOOI,SHN
```

17. Choose fifteen random integers from 65 to 90, inclusive, to serve as ASCII code numbers. Convert these fifteen integers to the fifteen characters for which they stand and print the results.

18. Write a program which produces a sentence containing twenty nonsense words. Each word can contain from two to five letters. Use random numbers and the ASCII code to produce the words.

```
]RUN
SIJWX UDMP OJV EXUQN MAQ XU EX PGDXS
ZCTY RLZNH RXO YRZIH TALYS TWXO OZU
EV EXVPD RHQ NTRGG DA

]
```

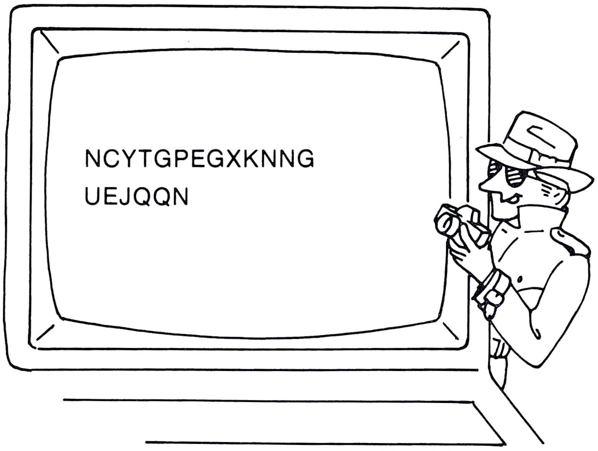
19. Using random numbers and the ASCII code numbers from 32 to 126, inclusive, have the computer generate a string of 100 characters, allowing repetitions. Tabulate how many characters are letters, numbers or miscellaneous characters. Print the results.

20. Using string functions, write a program to play a word guessing game. Ask the player to guess a secret word. Search through each guess to see if the guess contains any correct letters. If any letter is correct, have the computer print the ones that are. If the entire word is guessed, type "YOU GUESSED IT!!"

```
]RUN
Guess a word? HANGMAN
'A' is in the word.
'A' is in the word.
Guess a word? DATAPHONE
'A' is in the word.
'A' is in the word.
'E' is in the word.
Guess a word? TERMINAL
'E' is in the word.
'R' is in the word.
'A' is in the word.
'L' is in the word.
Guess a word? SCRABBLE
YOU GUESSED IT!!

]
```

21. You are a spy who will use the computer to produce a secret code.
(a) Input a short message and have the computer type back what appears to be nonsense words. To produce the coded words, convert each letter of the original message into its corresponding ASCII code number, add two to each number and convert to characters to produce the message. Keep all spaces between the words in their original places and understand that the letters Y and Z are to be converted to A and B.

(b) Write a program that will decode the following message:



NCYTGPEGXKNNG
UEJQQN

22. When the following program is run, the output is:

```
A = 1500          B = 999.999908
```

Explain in a clear, brief manner why the value of B is not exactly 1000, and why the value of A is exactly 1500.

```
10  FOR X = 1 TO 3000
20 A = A + 1 / 2
30 B = B + 1 / 3
40  NEXT X
50  PRINT "A=";A,"B=";B
60  END
```

23. Create bizarre "sentences" by having the computer print the words in a real sentence in reverse order. Your program should NOT reverse the order of the letters in each word.

24. Write a program which prints a triangle made up of parts of the word "triangle." The triangle should be obtuse, and the program should ask for its height (A), how far to indent the bottom edge (B), and where the bottom edge ends (C).

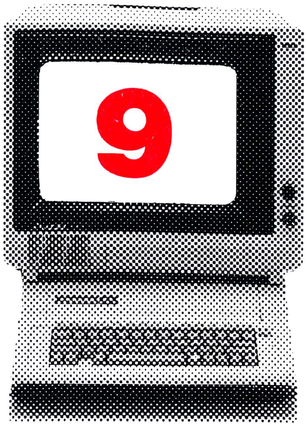
```
]RUN
A,B, AND C? 10,10,20
T
 RI
  IAN
   ANGL
    NGLET
     GLETRI
      LETRIAN
       ETRIANGL
        TRIANGLET
         RIANGLETRI

]
```

25. Enter a positive integer N$ as it would be expressed in binary form. That is, enter a string of ones and zeros. Have the computer print the equivalent of N$ as it is expressed in the decimal system.

01001010
01001001

26. Last night you were informed that your aunt had left you several million dollars. You decide to start your own corporation. In the tradition of DEC, GTE, RCA, IBM, and other great corporate conglomerates, you want your coporate name to be composed of as many initials as you like, each standing for a word (e.g., Radio Corporation of America becomes RCA). Write a program which accepts up to ten words and then prints a block of letters composed of the first letters of each word. For example, given “WE AWAIT SILENT TRYSTERO’S EMPIRE,” the program should return “WASTE.”

27. Often, literary critics argue over the true identity of the author of some ancient manuscript. To resolve such disputes, it is helpful to show similarities between the anonymous text and a text by a known author. Write a program which tabulates the occurrences of each article (“a”, “an”, and “the”), each adverb (check for “ly”), and of each mark of puunctuation (“.”, “,”, “!”, “?”, “;”, “:”).



FILES

OPEN
CLOSE
WRITE
READ
APPEND

OPEN
CLOSE
WRITE
READ
APPEND

So far we have discussed only two methods for saving data in the computer. Variables could be assigned a constant value (e.g., A = 5), or the data could be placed in DATA statements that are read by the computer when the program is run. There are two drawbacks to these methods. First, if it becomes necessary to update the data, program lines have to be retyped, thus altering the program and increasing the chance for errors. Second, since the data is an integral part of a program, it cannot be accessed by other programs. The solution to these problems is to store the data in a file. Since files are separate from programs, they can be updated without changing any part of a program. Also, a single data file can be accessed by many programs, allowing more flexible use of the data. Files and programs should not be confused. A program is a set of instructions that tell the computer what to do; a file is a set of data used by a program.

The computer uses two different types of files: sequential and random-access. As the name suggests, sequential files store and recall the information sequentially, that is, one item of data after another. Every time data is to be recalled from a sequential file, the computer must start from the beginning and scan the data in the file until the desired item or items have been found. Sequential files are best suited to situations in which the data will be recalled in the same order as it was originally stored. However, since accessing a file is time consuming, this is a very inefficient method for recalling information in a random order. Random-access files are better suited to this purpose.

The first part of this chapter will explain how to use sequential files; the second part will cover random-access files. Before learning the use of either sequential or random-access files, the programmer should first read Appendix B to become familiar with the Disk Operating System (DOS) and the disk drives.

Format of File Commands

When used within a program commands relating to the use of disk files must be sent to the Disk Operating System by a PRINT statement. In addition to the file command, the PRINT statement must also include a CTRL-D (Control-D) immediately preceding the file command. It is the presence of the CTRL-D that indicates to the computer that what follows is to be treated as a file command. For example,

```
10 PRINT ``CATALOG''
```

will produce a catalog of the programs and files stored on the disk in the drive when it is executed. However, since the computer does not display a CTRL-D on the screen as a printed character, the CTRL-D, which is between the first quotation mark and the C, is not visible. The exact purpose of the line is not clear upon inspection. Is it intended to produce a catalog of the disk or just to print the word CATALOG on the display? The following format is suggested to prevent the possiblity of confusion when reading programs:

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$;``CATALOG''
```

Defining DOS$ as CHR$(4) is the same as defining it to be a CTRL-D. (See Chapter 8.) This eliminates the possibility for confusion by making the presence of the CTRL-D obvious. The general format for all file commands used within a program should be

PRINT DOS$; “<command>”

with DOS$ having been defined previously as a CHR$(4).

Program 9.1

This program shows how a command is affected by the presence or absence of a CTRL-D:

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"CATALOG"
30  PRINT
40  PRINT "CATALOG": REM  No CTRL-D

]RUN


DISK VOLUME 254

*A 003 HELLO
*I 003 APPLESOFT
*B 006 LOADER.OBJ0
*B 042 FPBASIC
*B 042 INTBASIC
*A 003 MASTER
*B 009 MASTER CREATE
*I 009 COPY
*B 003 COPY.OBJ0
*A 009 COPYA
*B 003 CHAIN
*A 014 RENUMBER
*A 003 FILEM
*B 020 FID
*A 003 CONVERT13
*B 027 MUFFIN
*A 003 START13
*B 007 BOOT13
*A 004 SLOT#

CATALOG
```

OPEN

The OPEN command establishes a line of communication between a program and a file. The general format for the OPEN command is:

PRINT DOS$; “OPEN <file name>”

A name is used by the computer to identify each file and program stored on a disk. Therefore, each file and program stored on a disk must have a separate name. For example, the program fragment

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$; ''OPEN PAYROLL''
```

will instruct the computer to search for a file named PAYROLL on the disk in the drive. If it is not located, the computer will automatically create an empty file with that name so that data can be transferred from the computer's memory to the new file. If the file already exists, the computer can use it as a source of data.

CLOSE

Any file previously opened must be closed. This procedure is necessary in order to break the line of communication between a program in the computer's memory and a file. The format for the CLOSE command is:

```
PRINT DOS$; ''CLOSE <file name>''
```

For example,

```
50 PRINT DOS$; ''CLOSE PAYROLL''
```

will close the file PAYROLL.

Closing a file ensures that all of its information will be retained properly. A disk with open files on it should not be removed from the disk drive. Removing a disk prematurely may result in the loss of data.

WRITE

After the OPEN command establishes a line of communication between a program and a file, the WRITE command enables the user to place data in the file. Its form is:

```
PRINT DOS$; ''WRITE <file name>''
```

For example, the program fragment

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$; ''OPEN PAYROLL''
30 PRINT DOS$; ''WRITE PAYROLL''
```

will allow the user to place data in the file PAYROLL. After line 30 has been executed, subsequent PRINT statements will send data to the file PAYROLL instead of to the display until the file is closed, another file command is executed or the effect of the WRITE command is canceled by printing a CTRL-D (CHR$(4)) without a file command.

Program 9.2

This program, named CREATE, opens a file named WORK. In it are stored the names of four employees, their hourly wages and the number of hours they have worked. Note the structure of line 80, in which the name (N$), wage (W) and hours worked (H) for each employee are transferred to the file:

```
1  REM  N$ = Employee name
2  REM  W = Wage
3  REM  H = Hours worked
4  REM
5  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN WORK"
30  FOR X = 1 TO 4
40  INPUT "NAME? ";N$
50  INPUT "WAGE? ";W
60  INPUT "HOURS WORKED? ";H
70  PRINT DOS$;"WRITE WORK"
80  PRINT N$;",";W;",";H
90  PRINT DOS$
100  PRINT
110  NEXT X
120  PRINT DOS$;"CLOSE WORK"
130  PRINT "THE INFORMATION IS IN THE FILE."

]RUN
NAME? GWEN WALDEN
WAGE? 4.45
HOURS WORKED? 39

NAME? SANDY BECKERMAN
WAGE? 4.75
HOURS WORKED? 42

NAME? RUBEN STECK
WAGE? 5.10
HOURS WORKED? 36.75

NAME? RUTH OBRE
WAGE? 4.95
HOURS WORKED? 27.5

THE INFORMATION IS IN THE FILE.

]
```

The file WORK is opened at line 20. The values of N$, W and H for each employee are entered at lines 40 through 60. Line 70 instructs the computer to send the output of any subsequent PRINT statements to the file WORK instead of to the display. Line 80 actually transfers the contents of the variables N$, W and H to the file, but it has no effect on the display. Note its structure. If the contents of more than one variable are to be placed in a file by a single PRINT statement, commas within quotation marks must be placed between each variable.

The commas are stored in the file and act as markers so that the computer can distinguish each item of data when reading the file. The effect of the WRITE command is canceled at line 90, as demonstrated by the fact that the PRINT at line 100 produces a blank line on the display. If the WRITE command had not been canceled, the computer would have set the prompt from the INPUT at line 40 (NAME?) to the file instead of to the screen, thus ruining the data in the file. The effect of any file command, except CLOSE, can be canceled by printing a lone CTRL-D (CHR$(4)). After the data is sent to the file, it is closed at line 120.

Review

1. Write a program named CALENDAR that will create a file named MONTHS OF THE YEAR and store the names of the months of the year in it.

READ

The READ command enables the user to read data from a file, that is, transfer the data from the file to the computer's memory. Its form is:

```
PRINT DOS$; “READ <file name>”
```

For example, the program fragment

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$; “OPEN PAYROLL”
30 PRINT DOS$; “READ PAYROLL”
```

will allow the user to read data from the file PAYROLL. After line 30 has been executed, subsequent INPUT statements will take data from the file PAYROLL instead of from the keyboard until the file is closed, another file command is executed or the READ command is canceled with a CTRL-D.

Program 9.3

This program, named HOURS, will open the file WORK created by Program 9.2 and print the names and hours worked for the four employees:

```
1   REM   N$ = Employee name
2   REM   W = Wage
3   REM   H = Hours worked
4   REM
5   REM
10  DOS$ =  CHR$ (4): REM  CTRL-D
20   PRINT DOS$;"OPEN WORK"
30   PRINT DOS$;"READ WORK"
40   PRINT "NAME"; TAB( 20);"HOURS WORKED"
```

```
50  FOR X = 1 TO 4
60  INPUT N$,W,H
70  PRINT N$; TAB( 23);H
80  NEXT X
90  PRINT DOS$;"CLOSE WORK"

]RUN
NAME                  HOURS WORKED
GWEN WALDEN              39
SANDY BECKERMAN          42
RUBEN STECK              36.75
RUTH OBRE                27.5

]
```

The data in WORK FILE appears as follows:

GWEN WALDEN, 4.45, 39
SANDY BECKERMAN, 4.75, 42
RUBEN STECK, 5.1, 36.75
RUTH OBRE, 4.95, 27.5

The variable W is read in the input line but is not printed. The computer has an internal pointer that indicates the next item of information that is to be transferred from the file by the INPUT statement. When the INPUT statement is executed, the pointer is moved across the data line as each item is read. To ensure that the data in the file is read in the correct sequence, it is important that all of the necessary variables be included in the INPUT statement, even if some of them are not used by the program.

Review

2. Write a program named ODD MONTHS that will print the name of every other month of the year, taken from the file MONTHS OF THE YEAR created in Review problem 1.

```
]RUN
ODD MONTHS OF THE YEAR -
JANUARY
MARCH
MAY
JULY
SEPTEMBER
NOVEMBER

]
```

Updating Sequential Files, APPEND

Using the APPEND command instead of the OPEN command to open a file allows the user to add new information to the end of an existing file. When an OPEN command is executed, the computer opens the file and prepares to read from or write to the file starting at the beginning. If a programmer opens an old file with the OPEN command and then sends new data to the file, the new data will be written over old data, thus destroying it. To avoid this loss of data, use the APPEND command to instruct the computer to write new data at the end of the file. Its form is:

```
PRINT DOS$; “APPEND <file name>”
```

For example, the program fragment

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$; “APPEND PAYROLL”
30 PRINT DOS$: “WRITE PAYROLL”
```

will instruct the computer to add the information from any subsequent PRINT statements to the end of the file PAYROLL.

Program 9.4

This program, named ADD, will add information on two new employees just hired to the file WORK created by Program 9.2:

```
1   REM  N$ = Employee name
2   REM  W = Wage
3   REM  H = Hours worked
4   REM
5   REM
10  DOS$ =  CHR$ (4): REM  CTRL-D
20   PRINT DOS$;"APPEND WORK"
30   FOR X = 1 TO 2
40   INPUT "NAME? ";N$
50   INPUT "WAGE? ";W
60   INPUT "HOURS WORKED? ";H
70   PRINT DOS$;"WRITE WORK"
80   PRINT N$;",";W;",";H
90   PRINT DOS$
100  PRINT
110  NEXT X
120  PRINT DOS$;"CLOSE WORK"
130  PRINT "THE FILE HAS BEEN UPDATED."

]RUN
NAME? ROBERT LYNCH
WAGE? 4.60
HOURS WORKED? 44.25

NAME? ERMA MCCONNELL
WAGE? 4.80
HOURS WORKED? 41.5

THE FILE HAS BEEN UPDATED.

]
```

After changing the FOR . . . NEXT loop in Program 9.3 the user can rerun it to show that the new data is now in the file WORK.

```
]RUN
NAME                        HOURS WORKED
GWEN WALDEN                     39
SANDY BECKERMAN                 42
RUBEN STECK                     36.75
RUTH OBRE                       27.5
ROBERT LYNCH                    44.25
ERMA MCCONNELL                  41.5

]
```

There is no single command that will remove or alter outdated information in a sequential file. To change a sequential file, the programmer must transfer the information to be kept to a new file along with any new or corrected information. After the transfer the old file can be deleted and the new file given an appropriate name.

Program 9.5

This program, named RETIRE, will remove from the file WORK the name of an employee who has retired:

```
1   REM  N$ = Employee name
2   REM  W = Wage
3   REM  H = Hours worked
4   REM  R$ = Retired employee's name
5   REM
6   REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  INPUT "WHO HAS RETIRED? ";R$
30  PRINT DOS$;"OPEN WORK"
40  PRINT DOS$;"OPEN TEMPORARY"
50  FOR X = 1 TO 6
60  PRINT DOS$;"READ WORK"
70  INPUT N$,W,H
80  IF N$ = R$ THEN 110
90  PRINT DOS$;"WRITE TEMPORARY"
100  PRINT N$;",";W;",";H
110  NEXT X
120  PRINT DOS$;"CLOSE WORK"
130  PRINT DOS$;"CLOSE TEMPORARY"
140  PRINT DOS$;"DELETE WORK"
150  PRINT DOS$;"RENAME TEMPORARY,WORK"
160  PRINT "THE INFORMATION ON ";R$
170  PRINT "HAS BEEN REMOVED."

]RUN
WHO HAS RETIRED? ERMA MCCONNELL
THE INFORMATION ON ERMA MCCONNELL
HAS BEEN REMOVED.

]
```

The loop between lines 50 and 110 reads all of the data from the old file and transfers the desired data to the file TEMPORARY. After the data has been transferred, the old file WORK is deleted at line 140 and then the new file TEMPORARY is given the name WORK at line 150.

Program 9.5 works well in this limited example, but the technique it uses is not always practical. A FOR . . . NEXT loop was used to prevent the computer from attempting to read past the end of the file. However, this method could only be used because the length of the file WORK was known. To read a file of unknown length, use a loop containing a GOTO statement to ensure that the entire file is read. However, when the computer tries to read past the end of the file, an error will occur. This will stop the program, leaving the file open, possibly resulting in the loss of data. To prevent this, use an ONERR GOTO statement (covered in Chapter Five).

Program 9.6

This program, named RETIRE2, is a revision of Program 9.5, which allows the file WORK to be of any length:

```
1   REM  N$ = Employee name
2   REM  W = Wage
3   REM  H = Hours worked
4   REM  R$ = Retired employee's name
5   REM
6   REM
10  DOS$ =  CHR$ (4): REM  CTRL-D
20   ONERR  GOTO 200:T = 0
30   INPUT "WHO HAS RETIRED? ";R$
40   PRINT DOS$;"OPEN WORK"
50   PRINT DOS$;"OPEN TEMPORARY"
60   PRINT DOS$;"READ WORK"
70   INPUT N$,W,H
80   IF N$ = R$ THEN T =  - 1: GOTO 60
90   PRINT DOS$;"WRITE TEMPORARY"
100   PRINT N$;",";W;",";H
110   GOTO 60
120   REM
130   REM
200   REM  End of file
210   REM
220   PRINT DOS$;"CLOSE WORK"
230   PRINT DOS$;"CLOSE TEMPORARY"
240   IF T =  - 1 THEN 280
250   PRINT DOS$;"DELETE TEMPORARY"
260   PRINT "THERE IS NO ";R$;" IN THE FILE."
270   END
280   PRINT DOS$;"DELETE WORK"
290   PRINT DOS$;"RENAME TEMPORARY,WORK"
300   PRINT "THE INFORMATION ON ";R$
310   PRINT "HAS BEEN REMOVED."

]RUN
WHO HAS RETIRED? JEANNETTE MCDONALD
THERE IS NO JEANNETTE MCDONALD IN THE FILE.

]
```

Program 9.6 demonstrates two major improvements over Program 9.5. First, the program informs the user if a name has been entered that is not in the file, using the variable T as an indicator. If the name is in the file, T becomes -1; if not, T remains 0. Second, a GOTO loop with an error trap is used to read the file. This technique is recommended for reading most files.

Review

3. Write a program to remove the names of the even months of the year from the file created in Review problem 1.

Demonstration Programs

The following programs illustrate applications for the material covered in this part of the chapter. They consist of a series of examples concerning the record-keeping process of a small company:

Program 9.7

This program, named ACME, will produce a file named ADDRESSES that contains the names and addresses of all the people working for the Acme Sporting Goods Company. The company will use this file to mail their weekly paychecks:

```
1  REM  N$ = Employee's name
2  REM  S1$ = Street address
3  REM  C$ = City
4  REM  S2$ = State
5  REM  Z$ = ZIP Code
6  REM
7  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN ADDRESSES"
30  INPUT "HOW MANY EMPLOYEES ARE THERE? ";N
40  FOR E = 1 TO N
50  INPUT "NAME? ";N$
60  INPUT "STREET ADDRESS? ";S1$
70  INPUT "CITY, STATE? ";C$,S2$
80  INPUT "ZIP CODE? ";Z$
90  PRINT DOS$;"WRITE ADDRESSES"
100  PRINT N$;",";S1$;",";C$;",";S2$;",";Z$
110  PRINT DOS$
120  PRINT
130  NEXT E
140  PRINT DOS$;"CLOSE ADDRESSES"
150  PRINT "THE INFORMATION IS IN THE FILE."
```

```
]RUN
HOW MANY EMPLOYEES ARE THERE? 10
NAME? MARJORIE LIEBMAN
STREET ADDRESS? 23 PINE ST.
CITY, STATE? EDISON,NJ
ZIP CODE? 08442

NAME? HARRY STEIN
STREET ADDRESS? 15 ALLEN DR.
CITY, STATE? TRENTON,NJ
ZIP CODE? 08688

•
•
•
```

(A complete run is not shown)

```
NAME? ANNABEL LAFFIN
STREET ADDRESS? 14 ST. GEORGES AVE.
CITY, STATE? NEWTOWN,PA
ZIP CODE? 11448

THE INFORMATION IS IN THE FILE.

]
```

Program 9.8

The company saves money on mailing costs by providing the Post Office with pre-sorted mail. This program, named ACME2, prints the mailing labels for all the employees who live in a particular state:

```
 1  REM  N$ = Employee's name
 2  REM  S1$ = Street address
 3  REM  C$ = City
 4  REM  S2$ = State
 5  REM  Z$ = ZIP Code
 6  REM  TS$ = Target state
 7  REM
 8  REM
 10 DOS$ =  CHR$ (4): REM  CTRL-D
 20  INPUT "WHICH STATE DO YOU WANT? ";TS$
 30  ONERR  GOTO 200
 40  PRINT DOS$;"OPEN ADDRESSES"
 50  PRINT DOS$;"READ ADDRESSES"
 60  INPUT N$,S1$,C$,S2$,Z$
 70  IF S2$ = TS$ THEN 90
 80  GOTO 60
 90  PRINT : PRINT N$
 100  PRINT S1$: PRINT C$;",";S2$;"   ";Z$
 110  PRINT
```

```
120  GOTO 60
130  REM
140  REM
200  REM  End of file
210  REM
220  PRINT DOS$;"CLOSE ADDRESSES"

]RUN
WHICH STATE DO YOU WANT? PA

JERRY CLINTON
12 WESTCOTT RD.
YARDLEY,PA   11486


ANNABEL LAFFIN
14 ST. GEORGES AVE.
NEWTOWN,PA   11448


]
```

Program 9.9

This program, named ACME3, will allow the user to change the address of an employee who has moved:

```
1  REM  N$ = Employee's name
2  REM  S1$ = Street address
3  REM  C$ = City
4  REM  S2$ = State
5  REM  Z$ = ZIP Code
6  REM  M$ = Employee who moved
7  REM
8  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  INPUT "WHO HAS MOVED? ";M$
30  ONERR  GOTO 400
40  PRINT DOS$;"OPEN ADDRESSES"
50  PRINT DOS$;"OPEN TEMPORARY"
60  PRINT DOS$;"READ ADDRESSES"
70  INPUT N$,S1$,C$,S2$,Z$
80  IF N$ = M$ THEN 200
90  PRINT DOS$;"WRITE TEMPORARY"
100  PRINT N$;",";S1$;",";C$;",";S2$;",";Z$
110  GOTO 60
120  REM
130  REM
```

```
200  REM  Enter new address
210  REM
220  PRINT DOS$
230  PRINT : PRINT S1$
240  PRINT C$;",";S2$;"    ";Z$
250  PRINT : PRINT "ENTER NEW ADDRESS:"
260  INPUT "STREET ADDRESS? ";S1$
270  INPUT "CITY, STATE? ";C$,S2$
280  INPUT "ZIP CODE? ";Z$
290  PRINT
300  GOTO 90
310  REM
320  REM
400  REM  End of file
410  REM
420  PRINT DOS$;"CLOSE ADDRESSES"
430  PRINT DOS$;"CLOSE TEMPORARY"
440  PRINT DOS$;"DELETE ADDRESSES"
450  PRINT DOS$;"RENAME TEMPORARY,ADDRESSES"
460  PRINT M$;"'S ADDRESS HAS BEEN CHANGED."

]RUN
WHO HAS MOVED? MIKE SPRINGER

843 BRANDON RD.
ALLENTOWN,NJ    08904

ENTER NEW ADDRESS:
STREET ADDRESS? 17 HART AVE.
CITY, STATE? CRANBURY,NJ
ZIP CODE? 08263

MIKE SPRINGER'S ADDRESS HAS BEEN CHANGED.

]
```

Random-Access Files

A random-access file can be considered a collection of individual "sub-files" called records. The data in each record in a random-access file can be recalled, changed or erased entirely, in any order desired, without going through any of the other records in a file, i.e., the number of characters it can hold, is specified when a random-access file is created. The records in a random-access file can be as short or long as needed, but all of the records in a given file must be of the same length. A single record can hold as many strings and/or numbers as desired as long as the total number of characters does not exceed the capacity of the record. The only restriction is that each string may not be longer than 239 characters. The records in a random-access file are numbered sequentially from 1 to N, with N the number of records in the file.

OPEN

Random access files use a modified version of the OPEN command. Its form is:

```
PRINT DOS$; ''OPEN <filename>,L<record length>''
```

For example, the program fragment

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$;''OPEN INVENTORY,L50''
```

will open a random-access file named INVENTORY. Each record in the file will have a length of 50 characters. The length of a record must be at least equal to the number of characters to be stored in it plus one. Blank spaces, commas used to separate items of data and each digit in a number count as characters. For example, assuming

```
M$ = ''JUNE''
S$ = ''PEARL''
```

and

```
D = 22
```

then the statement

```
PRINT M$;'',''; D;'',''; S$
```

would send 13 characters to a file, so the records in that file would have to be at least 14 characters long to hold all the data properly. Records can be partially or even totally empty, but never try to put more characters in a record than it can hold. Doing so could adversely affect the data in adjacent records and could lead to confusion when recalling data from the file.

Every time a random-access file is opened, the record length must be specified. If the record length is not specified to be the same as when the file was first opened, it will not be possible to read old data from the file or transfer new data to it correctly. The computer does not keep a record of the correct record length for a file. Therefore, it will be necessary for the programmer to keep a careful record. The same procedure is used to close random-access files as is used to close sequential files.

WRITE

The form of the WRITE command used with random-access files is:

```
PRINT DOS$;''WRITE <file name>,R'';<record#>
```

For example, the program fragment

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$;‘‘OPEN INVENTORY,L50’’
30 Q = 4
40 PRINT DOS$;‘‘WRITE INVENTORY,R’’;Q
```

will instruct the computer to send data from subsequent PRINT statements to the fourth record in the file INVENTORY.

Program 9.10

This program, named ENTER, creates a file named INVENTORY that contains the product description and current inventory level for each of the products sold by a small mail order catalog company. A product's catalog number corresponds to the record in the file used to hold the information on that product:

```
1   REM  PD$ = Product description
2   REM  I = Current inventory level
3   REM  C = Catalog number
4   REM
5   REM
10  DOS$ =  CHR$ (4): REM  CTRL-D
20   PRINT DOS$;"OPEN INVENTORY,L35"
30   FOR C = 1 TO 25
40   PRINT "CATALOG NUMBER ";C;":"
50   INPUT "DESCRIPTION? ";PD$
60   INPUT "INVENTORY? ";I
70   PRINT DOS$;"WRITE INVENTORY,R";C
80   PRINT PD$;",";I
90   PRINT DOS$
100   PRINT
110   NEXT C
120   PRINT DOS$;"CLOSE INVENTORY"

]RUN
CATALOG NUMBER 1:
DESCRIPTION? ALLEN WRENCH
INVENTORY? 450

CATALOG NUMBER 2:
DESCRIPTION? SOCKET WRENCH
INVENTORY? 670



CATALOG NUMBER 25:
DESCRIPTION? CIGARETTE LIGHTER
INVENTORY? 386


]
```

It is assumed that the product description will not exceed 30 characters and that the inventory will never exceed 999, so the file is opened with a record length of 35 at line 20. (Remember to allow a character for the comma separating the description and inventory level.) The PRINT statement at line 80 sends the data on the product to the record corresponding to the catalog number of the product, as specified by the WRITE command at line 70.

Review

4. Rewrite the answer to Review problem 1, using random-access files.

READ

The form of the READ command used with random-access files is:

```
PRINT DOS$; ''READ <file name>, R''; <record#>
```

For example, the program fragment

```
10 DOS$ = CHR$(4) : REM CTRL-D
20 PRINT DOS$; ''OPEN INVENTORY, L50''
30 Q = 7
40 PRINT DOS$; ''READ INVENTORY, R''; Q
```

will instruct the computer to take data from subsequent INPUT statements from the seventh record of the file INVENTORY.

Program 9.11

This program, named RETRIEVE, prints the product description and current inventory level for a product, as determined by the catalog number entered:

```
1  REM  PD$ = Product description
2  REM  I = Current inventory level
3  REM  C = Catalog number
4  REM
5  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  ONERR  GOTO 300
30  PRINT "ENTER A NEGATIVE NUMBER TO STOP"
40  PRINT DOS$;"OPEN INVENTORY,L35"
50  INPUT "CATALOG NUMBER? ";C
60  IF C < 0 THEN 200
70  PRINT DOS$;"READ INVENTORY,R";C
80  INPUT PD$,I
90  PRINT "DESCRIPTION: ";PD$
100  PRINT "CURRENT INVENTORY: ";I
110  PRINT
120  PRINT DOS$
130  GOTO 50
140  REM
150  REM
200  REM  End program
210  PRINT DOS$;"CLOSE INVENTORY"
220  END
230  REM
240  REM
300  REM  Error on attempting to read past end of file
310  REM
320  PRINT "INVALID CATALOG NUMBER"
330  PRINT "PLEASE REENTER"
340  GOTO 40

]RUN
ENTER A NEGATIVE NUMBER TO STOP
CATALOG NUMBER? 9
DESCRIPTION: SPARK PLUGS
CURRENT INVENTORY: 190

CATALOG NUMBER? 2
DESCRIPTION: SOCKET WRENCH
CURRENT INVENTORY: 670

CATALOG NUMBER? 27
INVALID CATALOG NUMBER
PLEASE REENTER
CATALOG NUMBER? 10
DESCRIPTION: SPARK PLUG GAPPER
CURRENT INVENTORY: 539

CATALOG NUMBER? -1

]
```

Note the use of the error trap to determine invalid catalog numbers when a number greater than the number of records in the file is entered. The GOTO is used at line 340 to send program execution back to line 40 so that the file can be reopened after the error, which closed the file. A resume would have sent program execution back to line 80, where the error occurred.

Review

5. Rewrite the answer to Review problem 2, using a random-access file.

Demonstration Programs

The following programs illustrate applications for the material covered in this part of the chapter. They consist of a series of examples concerning the record-keeping process of a small bank:

Program 9.12

This program, named CREATE ACCOUNTS, will create a random-access file named ACCOUNTS that will contain the customer name and current balance for the fifteen savings accounts on deposit at the Three R's Savings and Loan Association:

```
1  REM  N$ = Account holder's name
2  REM  B = Account balance
3  REM  A = Account number
4  REM
5  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN ACCOUNTS,L30"
30  FOR A = 1 TO 15
40  PRINT "THIS WILL BE ACCOUNT #";A
50  INPUT "NAME? ";N$
60  INPUT "BALANCE? ";B
70  PRINT DOS$;"WRITE ACCOUNTS,R";A
80  PRINT N$;",";B
90  PRINT DOS$
100  PRINT
110  NEXT A
120  PRINT DOS$;"CLOSE ACCOUNTS"

]RUN
THIS WILL BE ACCOUNT #1
NAME? WILLIAM KRIEG
BALANCE? 300

THIS WILL BE ACCOUNT #2
NAME? ELLEN LAYTON
BALANCE? 504


THIS WILL BE ACCOUNT #15
NAME? BRIAN WILSON
BALANCE? 489.31


]
```

Program 9.13

This program, named UPDATE ACCOUNTS, will update the data in the file ACCOUNTS whenever an individual makes a deposit or withdrawal:

```
1  REM  N$ = Account holder's name
2  REM  B = Account balance
3  REM  A = Account number
4  REM  T$ = Transaction type
5  REM  T = Amount of transaction
6  REM
7  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  ONERR  GOTO 300
30  PRINT "ENTER A NEGATIVE NUMBER TO STOP"
40  PRINT DOS$;"OPEN ACCOUNTS,L30"
50  INPUT "WHICH ACCOUNT? ";A
60  IF A < 0 THEN 200
70  PRINT DOS$;"READ ACCOUNTS,R";A
80  INPUT N$,B
90  PRINT DOS$
100  PRINT "OLD BALANCE: ";B
110  INPUT "DEPOSIT (D) OR WITHDRAWAL (W)? ";T$
120  INPUT "AMOUNT? ";T
130  IF T$ = "W" THEN T =  - T
140 B = B + T
150  PRINT "NEW BALANCE: ";B
160  PRINT DOS$;"WRITE ACCOUNTS,R";A
170  PRINT N$;",";B
180  PRINT DOS$
190  PRINT : GOTO 50
200  PRINT DOS$;"CLOSE ACCOUNTS"
210  END
220  REM
230  REM
300  REM  Error on attempting to read past end of file
310  REM
320  PRINT "INVALID ACCOUNT NUMBER"
330  PRINT "PLEASE REENTER"
340  GOTO 40
```

```
]RUN
ENTER A NEGATIVE NUMBER TO STOP
WHICH ACCOUNT? 8
OLD BALANCE: 555.22
DEPOSIT (D) OR WITHDRAWAL (W)? W
AMOUNT? 317.49
NEW BALANCE: 237.73

WHICH ACCOUNT? 4
OLD BALANCE: 287.64
DEPOSIT (D) OR WITHDRAWAL (W)? D
AMOUNT? 104.50
NEW BALANCE: 392.14
```

```
WHICH ACCOUNT? 11
OLD BALANCE: 67.59
DEPOSIT (D) OR WITHDRAWAL (W)? W
AMOUNT? 212.95
NEW BALANCE: -145.36

WHICH ACCOUNT? -1

]
```

When updating a record in a random-access file, read the contents of the entire record from the file so that the old information that is to be saved can be returned to the record along with the new information in the correct order. In this case the account holder's name is read from the file at line 80 so that it can be returned to the file along with the new balance at line 170.

Program 9.14

This program, named WRITE LETTERS, uses the file ACCOUNTS to send a letter of warning to the holders of all overdrawn accounts or a letter of congratulations to the holders of accounts with $500 or more, informing them that they will be receiving a toaster in the mail:

```
1  REM  N$ = Account holder's name
2  REM  B = Account balance
3  REM  A = Account number
4  REM
5  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN ACCOUNTS,L30"
30  FOR A = 1 TO 15
40  PRINT DOS$;"READ ACCOUNTS,R";A
50  INPUT N$,B
60  PRINT DOS$
70  IF B < 0 THEN 200
80  IF B >  = 500 THEN 400
90  NEXT A
100  PRINT DOS$;"CLOSE ACCOUNTS"
110  END
120  REM
130  REM
200  REM  Overdraft letter
210  REM
220  PRINT : PRINT "DEAR ";N$;","
230  PRINT "YOUR ACCOUNT IS OVERDRAWN BY"
240  PRINT "$"; - B;".  PLEASE DEPOSIT THAT"
250  PRINT "AMOUNT IMMEDIATELY."
260  PRINT
270  PRINT  TAB( 15);"THANK YOU,"
280  PRINT  TAB( 15);"THREE R'S S & L"
290  PRINT : PRINT
300  GOTO 90
```

```
310  REM
320  REM
400  REM  Award letter
410  REM
420  PRINT : PRINT "DEAR ";N$;","
430  PRINT "YOU ARE ONE OF OUR LARGEST"
440  PRINT "DEPOSITORS.  YOUR COMPLIMENTARY"
450  PRINT "TOASTER IS IN THE MAIL."
460  PRINT
470  PRINT  TAB( 15);"CONGRATULATIONS,"
480  PRINT  TAB( 15);"THREE R'S S & L"
490  PRINT : PRINT
500  GOTO 90
```

```
]RUN

DEAR ELLEN LAYTON,
YOU ARE ONE OF OUR LARGEST
DEPOSITORS.  YOUR COMPLIMENTARY
TOASTER IS IN THE MAIL.

              CONGRATULATIONS,
              THREE R'S S & L



DEAR JANE LAWSON,
YOU ARE ONE OF OUR LARGEST
DEPOSITORS.  YOUR COMPLIMENTARY
TOASTER IS IN THE MAIL.

              CONGRATULATIONS,
              THREE R'S S & L



DEAR IDA HILLS,
YOUR ACCOUNT IS OVERDRAWN BY
$145.36.  PLEASE DEPOSIT THAT
AMOUNT IMMEDIATELY.

              THANK YOU,
              THREE R'S S & L



DEAR GEORGE DICKEL,
YOU ARE ONE OF OUR LARGEST
DEPOSITORS.  YOUR COMPLIMENTARY
TOASTER IS IN THE MAIL.

              CONGRATULATIONS,
              THREE R'S S & L
```

Multiple File Use

It is possible to work with two or more open files within a program, as shown by the following two programs. When this is done, the programmer should not confuse the variables associated with each file opened by a program.

Program 9.15

This program, named CREATE STUDENT FILES, will create two random-access files. One, called NAMES, will contain the names of the fifteen students attending a small school. The student's name will be stored in the record in the file corresponding to the student's ID number. The second file, called MAJORS, will contain the elected major of each student in the school. Like NAMES, the contents of MAJORS will also be referenced by the students' ID numbers:

```
1   REM  N$ = Student's name
2   REM  M$ = Student's major
3   REM  S = Student's ID number
4   REM
5   REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN NAMES,L25"
30  PRINT DOS$;"OPEN MAJORS,L20"
40  FOR S = 1 TO 15
50  INPUT "NAME? ";N$
60  PRINT N$;" HAS BEEN ASSIGNED ID# ";S
70  INPUT "MAJOR? ";M$
80  PRINT DOS$;"WRITE NAMES,R";S
90  PRINT N$
100  PRINT DOS$;"WRITE MAJORS,R";S
110  PRINT M$
120  PRINT DOS$
130  PRINT
140  NEXT S
150  PRINT DOS$;"CLOSE NAMES"
160  PRINT DOS$;"CLOSE MAJORS"
```

```
]RUN
NAME? GWEN WALDEN
GWEN WALDEN HAS BEEN ASSIGNED ID# 1
MAJOR? POLITICAL SCIENCE

NAME? RUTH OBRE
RUTH OBRE HAS BEEN ASSIGNED ID# 2
MAJOR? TRAVEL & TOURISM
```

Program 9.16

This program, named MAJOR REPORT, will print the names of all the students with a particular major:

```
1  REM  N$ = Student's name
2  REM  M$ = Student's major
3  REM  S = Student's ID number
4  REM  TM$ = Target major
5  REM
6  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN NAMES,L25"
30  PRINT DOS$;"OPEN MAJORS,L20"
40  INPUT "WHICH MAJOR? ";TM$
50  FOR S = 1 TO 15
60  PRINT DOS$;"READ MAJORS,R";S
70  INPUT M$
80  IF M$ = TM$ THEN 200
90  NEXT S
100  PRINT DOS$;"CLOSE NAMES"
110  PRINT DOS$;"CLOSE MAJORS"
120  END
130  REM
140  REM
200  REM  Major matches, so get name
210  REM
220  PRINT DOS$;"READ NAMES,R";S
230  INPUT N$
240  PRINT N$
250  GOTO 90

]RUN
WHICH MAJOR? TRAVEL & TOURISM
RUTH OBRE
JOHN ORR
SAM HILL

]
```

Exercises

1. Store 50 random numbers between 0 and 20 in a sequential file. Use a second program to retrieve the numbers, add them, and print the sum.

2. Store in a sequential file the names and prices of five different desserts served at MADGE'S DINER. With a second program add two additional desserts. Have a third program retrieve and print the information.

3. Store ten different first names of friends in a sequential file. A second program is to print all the names in the file which begin with the letters D, E, F, G, and H or a message if none are found.

4. Store the 26 letters of the alphabet in a random-access file in order. Have a program pick five random numbers from 1 to 26 and use these numbers to put together a five letter "word."

5. Store titles of ten books in a random-access file. Use this file to make a sequential file for all the titles beginning with letters from N to Z, inclusive.

6. (a) Create a random-access file named ACCOUNTS that will contain the customer name and current balance for twenty-five savings accounts.
   (b) Write a program that will update the file ACCOUNTS whenever an individual makes a deposit or withdrawal.
   (c) Write a program using the file ACCOUNTS that sends a letter of warning to the holders of all overdrawn accounts or a letter of congratulations to the holders of accounts with $500 or more informing them that they will be receiving a toaster in the mail.

7. (a) Create a random-access file named CARS to record how many full-sized, mid-sized and compact cars a dealer sells each month for a twelve-month period.
   (b) Write a program that will retrieve information from the file CARS for a specific month and then print a bar graph comparing the sales of the three sizes of cars.

8. (a) Write a program that will create a random-access file FRAT which contains the names, fraternities, and ages of thirty college students.
   (b) Write a program that will access FRAT and create a sequential file SIGMA which

contains the names and ages of the students who live in Sigma Chi.

(c) Write a program that accesses the file FRAT and randomly selects twenty-five students for seats in a classroom of five rows, five seats in a row. Have the computer print a seating plan for the class, placing each student's name at the correct seat location.

9. (a) Two persons, Sheryl and Carol, measured the Fahrenheit temperature (F) outside on February 12 at various times (T) during a ten-hour period. Their results are recorded here. Set up two sequential files, one for each person's data, naming each file after that person.

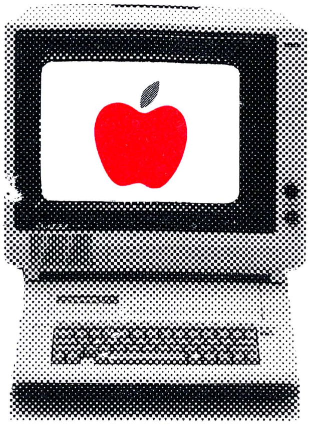
Sheryl		**Carol**	
T	**F**	**T**	**F**
0.0	18.1	1.0	20.9
2.1	24.0	1.9	23.3
3.8	27.2	3.5	26.1
6.0	29.3	6.0	28.8
8.0	26.6	8.2	26.2
9.0	16.1	10.0	16.0

(b) Write a program that will merge the two files into one sequential file named MERGE. The times should be sequentially in order. However, when a value of (T) occurs both in Sheryl and Carol, the average of the two values of (F) should be placed in MERGE.

(c) Retrieve and print the contents of MERGE.

10. (a) Create a sequential payroll file called PAY to store each of ten person's names (last name first), his or her hourly pay rate, the number of dependents, and deductions for medical and life insurance for each week. Supply appropriate data.

(b) Use PAY to prepare the payroll data sheet (supply the number of hours (H) each person worked during the week). The sheet should list the name, hours worked, gross pay, three deductions, and net pay. Assume a tax rate of 25%, with 2% being subtracted from this rate for each dependent.



APPENDICES

SPEED

SAVE

LOAD

STORE

RECALL

INIT

CATALOG

RENAME

DELETE

LOCK

UNLOCK

MAXFILES

APPENDIX A
EDITING

The Apple has several functions that can make editing and debugging a program easier. An outline of these features is presented here:

← and →

The left-arrow (marked ←) and right-arrow (marked →) keys can be used to correct an error in a line as the line is being typed. The ← key is used to back up the cursor to the location of the error so that it can be corrected by typing the correct symbol. The → key is used to move the cursor forward to the end of the line so that typing can be resumed or the RETURN key can be struck. This is necessary to ensure that the computer recognize the entire line entered. If the RETURN key is struck in the middle of a line, the computer erases all characters starting at the current location of the cursor and continuing to the right.

CONTROL-X

Typing the X key while depressing the CONTROL key instructs the computer to ignore the line currently being typed. A backslash (\) appears and the cursor moves to the next line:

```
]10 FOR X = 1 TO 5

]20 PRINT X+8

]20 NEXT X\
]30 NEXT X

]LIST

10  FOR X = 1 TO 5
20  PRINT X + 8
30  NEXT X

]
```

ESC A, B, C or D

The ESC key can be used in conjunction with the A, B, C and D keys to move the cursor around the screen. Pressing and releasing the ESC key and then pressing the A key moves the cursor one space to the right. Pressing the ESC key and then B moves the cursor one space to the left. ESC and C and ESC and D move the cursor one space down and one space up respectively.

CONTROL-S

Typing the S key while depressing the CONTROL key causes the computer to stop a program listing. Typing any other key causes the computer to continue.

SPEED

The SPEED command is used to control how fast the computer prints information, such as program output or listings, on the display. Its format is:

SPEED = <s>

where s is a number from 0 to 255, inclusive. For example,

SPEED = 100

will set the Apple's printing speed at about that of a fast typist, and the computer will print at that rate until the rate is changed by another speed command. The Apple is automatically set to the fastest speed, 255, when it is turned on.

Starting Programs at Arbitrary Locations

Including a line number after a RUN command will cause the computer to start running the program in memory at the specified line instead of at the beginning:

```
10  PRINT "AT 10"
20  PRINT "AT 20"

]RUN
AT 10
AT 20

]
```

APPENDIX B
CASSETTE OPERATION

A tape recorder can be connected to the Apple to allow programs and data to be saved and then recalled at a later time.

Saving a Program, SAVE

To save a program in the computer's memory on tape, start the tape recorder in the record mode and type SAVE. The Apple will beep to indicate that it has started to record on the tape. When the program has been stored, the computer will beep again. At this point the tape recorder can be turned off.

The user should keep a written record of the programs stored on a tape, along with their locations on the tape so that the tape may be correctly positioned when the user wishes to recall a program or file.

Recalling a Program, LOAD

To recall a program that has been stored on tape, position the tape to the beginning of the section in which the program has been stored. Start the tape recorder in the playback mode and type LOAD. The Apple will beep to indicate that it has started to recall the program. When the entire program has been recalled, the computer will beep again, indicating that the program is ready to be run. At this point the tape recorder can be turned off.

Storing Data, STORE

The Apple can store data on tape in the form of numeric subscripted variables. Once a subscripted variable has been dimensioned and the individual elements have been assigned their values, the command STORE will record the information on tape:

```
10  DIM A(15)
20  FOR B = 1 TO 15
30 A(B) = B ^ 2
40  NEXT B
50  INPUT "TURN ON THE TAPE, THEN HIT RETURN;"Z$
60  STORE A
```

Note that at line 50 the program employs the dummy variable Z$ to allow sufficient time to turn on the tape recorder and that line 60 stores all 15 elements of the subscriped variable A. See Chapter 4 for a discussion of subscripted variables.

Recalling Data, RECALL

The contents of a numeric subscripted variable which has been previously stored on tape can be retrieved with the RECALL command. The tape recorder must be in the playback mode and the variable must have been previously dimensioned to the appropriate size before the RECALL command is executed. It is important that the recalled subscripted variable be dimensioned exactly as the stored subscripted variable. Otherwise the data will not be read properly.

```
10   DIM A(15)
20   INPUT "TURN ON THE TAPE, THEN HIT RETURN";Z$
30   RECALL A
40   FOR Q = 1 TO 15
50   PRINT A(Q);"   ";
60   NEXT Q
```

Storing String on Tape

STORE and RECALL cannot be used with strings directly. Each character in the string must be converted to a numeric equivalent using the ASC function and then stored in a numeric subscripted variable. The following two programs will store and retrieve a string up to 255 characters long:

```
10   INPUT A$
20   DIM A(255)
30   FOR Q = 1 TO  LEN (A$)
40 A(Q) =  ASC ( MID$ (A$,Q,1))
50   NEXT Q
60 A(0) =  LEN (A$)
70   INPUT "TURN ON THE TAPE, THEN HIT RETURN";Z$
80   STORE A
```

```
10   DIM A(255)
20   INPUT "TURN ON THE TAPE, THEN HIT RETURN";Z$
30   RECALL A
40   FOR W = 1 TO A(0)
50 A$ = A$ +  CHR$ (A(W))
60   NEXT W
70   PRINT A$
```

Note that A(0) is used to store the length of the string. See Chapter 8 for a discussion of ASC, CHR$ and LEN.

APPENDIX C
DISKETTE OPERATION

Proper Care of Equipment

Diskettes should not be stored near any device which produces heat or a strong magnetic field such as a television set or computer monitor.

Making a New Diskette Operational, INIT

It is necessary to initialize, or "format," a new diskette before the computer can use it for the first time. First, the Disk Operating System (DOS) must be loaded into the computer from some previously initialized diskette, such as the System Master provided by the manufacturer. This is done by inserting the diskette in drive 1 and turning on the computer. The diskette to be initialized should then be substituted for the diskette in the drive. Type NEW and then enter a "greeting program." A greeting program is a program stored on the disk that the computer will run automatically when it is turned on with the diskette in the drive. A short example is shown here:

```
10  REM  Greating program
20  PRINT "HOMEWORK DISKETTE"
30  PRINT "INITIALIZED BY"
40  PRINT "GWEN WALDEN"
50  PRINT "FEBRUARY 17, 1984"
60  END
```

After the greeting program has been entered, the INIT command is used to initialize the diskette and save the greeting program on it. Its format is:

```
INIT <program name>
```

The greeting program will be saved under the name specified. For example,

```
INIT HELLO
```

will instruct the computer to initialize the diskette in the drive and store the program currently in memory on the diskette with the name HELLO.

SAVE

The SAVE command is used to store a program on a diskette. Its format is:

```
SAVE <program name>
```

For example,

SAVE TAXES

will save the program currently in memory on the diskette and give it the name TAXES. If there is already a program on the diskette with that name, the new program will replace it, thereby destroying the original.

The program name may be up to thirty characters long. The first character may not be a digit.

LOAD

Programs previously stored on a diskette can be recalled using the LOAD command. Its format is:

LOAD <program name>

For example

LOAD TAXES

will load the program TAXES into memory from the diskette. Any program previously in memory will be replaced. It is important to note that this procedure does not remove the program from the diskette. An exact copy of the program is made and placed in the computer's memory.

One program can load another program into memory. For example,

```
80 PRINT CHR$(4);''LOAD TAXES''
```

will load the program TAXES into memory, replacing the program that loaded it, when that line is executed. The significance of the CHR$(4) in this line is explained in Chapter Nine.

CATALOG

Typing the command CATALOG in immediate mode produces a list of all the programs and files on a diskette:

```
]CATALOG

DISK VOLUME 254

*A 002 HELLO
 B 034 PAGE2
 B 034 PAGE1
 B 010 TEXTGR
*B 010 TEXTSCREEN
 B 006 TEXTSCREEN2
 A 004 SOUND EXPERIMENT
 I 002 ROOT TABLES

]
```

The catalog is displayed in four columns. If the first column contains an asterisk, the program or file has been locked to prevent accidental removal. If no asterisk is present, it is unlocked. (A description of the commands LOCK and UNLOCK is presented later in this appendix.) The second column contains a single letter identifying the type of program or file listed. An A signifies a program written in Applesoft. An I signifies a program written in Integer BASIC. A T signifies a data file, and a B signifies a file or program in machine code. The number in the third column indicates the number of sectors the program or file takes up on the diskette. The minimum is two. The fourth column contains the name of the program or file.

The CATALOG command can also be used from within a program. For example, the line

```
65 PRINT CHR$(4);''CATALOG''
```

will produce a catalogue of the diskette in the drive when it is executed as part of a program.

RENAME

The RENAME command is used to change the name of a program or file that has been stored on a diskette. Its format is:

```
RENAME <old name>, <new name>
```

For example,

```
RENAME TAXES,IRS
```

will give the program TAXES on the diskette the new name IRS.

As with the LOAD and CATALOG commands, the RENAME command can also be used in a program. The following line, when executed as part of a program, will have the same effect as the immediate mode example shown above:

```
120 PRINT CHR$(4);''RENAME TAXES,IRS''
```

DELETE

Unwanted programs and files are removed from a diskette with the DELETE command. Its format is:

```
DELETE <program or file name>
```

For example,

```
DELETE IRS
```

will remove the program IRS from the diskette in the drive.

The DELETE command can also be used in a program. The following line will have the same effect as the immediate mode example shown above when it is executed as part of a program:

```
75 PRINT CHR$(4);"DELETE IRS"
```

LOCK and UNLOCK

The LOCK command is used to prevent a program or file from being removed accidentally from a diskette. Its format is:

```
LOCK <program or file name>
```

For example,

```
LOCK TAXES
```

will lock the program TAXES stored on the diskette. A subsequent attempt to rename or delete it will result in the error message FILE LOCKED.

Locked programs or files can be unlocked with the command UNLOCK. Its format is:

```
UNLOCK <program or file name>
```

For example:

```
UNLOCK TAXES
```

Once a program or file has been unlocked, it can be renamed or deleted.

MAXFILES

Normally the Apple will allow the user to have up to three files open simultaneously. This limit can be changed with the command MAXFILES. Its format is

```
MAXFILES <n>
```

where n is the maximum number of files (1 to 16 inclusive) the user wishes to have open simultaneously. For example, the command

```
MAXFILES 7
```

will allow up to seven files to be opened simultaneously.

Multiple Drive Systems

When you are using a system that has more than one diskette drive, it is often necessary to specify which drive you want to use. This is done by using an expanded syntax for the commands in this appendix (except MAXFILES). The expanded format is

```
<command>,D<d>
```

where d is the number of the desired drive. For example,

```
CATALOG,D2
```

will produce a list of the programs and files stored on the diskette in drive 2. Similarly,

```
SAVE TAXES,D1
```

will instruct the computer to save the program currently in memory under the name TAXES on the diskette in drive 1. When no drive is explicitly specified with a command, the computer will use the drive that was last specified with a command. When the computer is first turned on, it uses drive 1 until otherwise specified.

Review Answers

CHAPTER ONE

1.
```
10 X = 5
20 Y = 5 * X + 7
30  PRINT Y
```

2.
```
10 H$ = "HARRY"
20 S$ = "SHERRY"
30  PRINT "HELLO, ";H$
40  PRINT S$;" IS LOOKING FOR YOU."
```

3.
```
10  READ X
20 Y = 3 * X + 5
30  PRINT Y;" ";
40  GOTO 10
50  DATA  3,5,12,17,8
```

4.
```
5  PRINT "NAME","FIRST GRADE"
 10  READ N$,A,B,C,D
20  PRINT N$,A
30  GOTO 10
50  DATA  WATERS,83,95,86,80,KANE,56,97,66,89,MIKAN,61,83,42,90
```

5.
```
10  INPUT "WHAT IS X? ";X
20  PRINT "5*X = ";5 * X
30  PRINT "X/5 = ";X / 5
40  PRINT
50  GOTO 10
```

6.

```
10  INPUT "WHAT IS YOUR NAME? ";N$
20  INPUT "WHAT IS YOUR FRIEND'S NAME? ";F$
30  PRINT F$;" IS A FRIEND OF ";N$
40  PRINT
50  GOTO 10
```

CHAPTER TWO

1.

```
10  INPUT "Enter two numbers? ";A,B
20  IF A > B THEN  PRINT A: PRINT B: GOTO 10
30  PRINT B: PRINT A: GOTO 10
```

2.

```
10  INPUT "ENTER TWO LAST NAMES. ";N1$,N2$
20  IF N1$ > N2$ THEN  PRINT N2$: PRINT N1$: GOTO 10
30  PRINT N1$: PRINT N2$: GOTO 10
```

3.

```
10  INPUT "ENTER  A NUMBER. ";N
20  IF N > 25 AND N < 112 THEN 40
30  PRINT N;" IS OUT OF RANGE": GOTO 10
40  PRINT N;" IS BETWEEN 25 AND 112": GOTO 10
```

4.

```
10  INPUT "Enter N$ ";N$
20  IF N$ < "Garbage" OR N$ > "Trash" THEN 40
30  PRINT "NO": GOTO 10
40  PRINT "YES": GOTO 10
```

5.

```
10  FOR I = 1 TO 25
20  PRINT I;" ";
30  NEXT I
```

6.

```
10  FOR I = 20 TO 10 STEP  - 2
20  PRINT I;"  ";
30  NEXT I
40  PRINT
50  END
```

7.

```
10  INPUT "STEP VALUE? ";N
20  FOR I = 8 TO 20 STEP N
30  PRINT I;" ";
40  NEXT I
```

CHAPTER THREE

1.

```
10 R1 =  INT (101 *  RND (1) + 50)
20 R2 =  INT (101 *  RND (1) + 50)
30  PRINT R1;" MULTIPLIED BY ";R2;" IS ";R1 * R2
```

2.

```
10  PRINT "I'M THINKING OF A NUMBER FROM 1 TO 50."
20 R =  INT (50 *  RND (1) + 1)
30  FOR I = 1 TO 5
40  INPUT "WHAT IS YOUR GUESS? ";G
50  IF G < R THEN  PRINT "TOO LOW!": GOTO 80
60  IF G > R THEN  PRINT "TOO HIGH!": GOTO 80
70  PRINT "THAT IS CORRECT!!!": END
80  NEXT I
90  PRINT "YOU'VE HAD FIVE GUESSES NOW."
100  PRINT "THE NUMBER WAS ";R: END
```

3.

```
10  FOR I = 1 TO 50
20 R =  INT (9 *  RND (1))
30  IF R < 5 THEN L = L + 1
40  NEXT I
50  PRINT "THERE WERE ";L;" NUMBERS FROM 0 TO 4."
60  PRINT "THERE WERE ";50 - L;" NUMBERS FROM 5 TO 9."
```

CHAPTER FOUR

1.

```
10  FOR I = 20 TO 24
20  PRINT "OUTER: ";I
30  FOR J = 1 TO 3
40  PRINT "INNER:";J,
50  NEXT J
60  NEXT I
```

2.

```
10  FOR I = 1 TO 3
20  INPUT A(I)
30  NEXT I
40  FOR I = 3 TO 1 STEP  - 1
50  PRINT A(I)
60  NEXT I
```

3.

```
10  REM  read 6 words from the keyboard
20  FOR I = 1 TO 6
30  INPUT W$(I)
40  NEXT I
50  REM  pick 4 words and print them as a sentence
60  FOR I = 1 TO 4
70  PRINT W$( INT (6 *  RND (1) + 1));" ";
80  NEXT I
90  PRINT ".": REM  place period at end of sentence
```

4.

```
10  REM  read 6 words from the keyboard
20  FOR I = 1 TO 6
30  INPUT W$(I)
40  NEXT I
50  REM  pick 4 words and print them as a sentence
60 C(1) =  INT (6 *  RND (1) + 1)
70  PRINT W$(C(1));
80  FOR P = 2 TO 4
90 C(P) =  INT (6 *  RND (1) + 1)
100  REM  now check for duplicate random numbers
110  FOR K = 1 TO P - 1
120  IF C(K) = C(P) THEN 90: REM  pick another if used
130  NEXT K
140  PRINT " ";W$(C(P));
150  NEXT P
160  PRINT ".": REM  place period at end of sentence
170  END
```

5.

```
10  FOR I = 1 TO 6
20  INPUT X(I)
30  NEXT I
40  FOR I = 1 TO 6
50  PRINT X(I)
60  NEXT I
70  PRINT
80  FOR I = 1 TO 6
90  PRINT X(I);" ";
100  NEXT I
```

6.

```
10  DIM X$(5,3)
20  FOR I = 1 TO 5
30  FOR J = 1 TO 3
40  READ X$(I,J)
50  NEXT J
60  NEXT I
70  REM  output X$() making the rows become columns
80  FOR I = 1 TO 3
90  FOR J = 1 TO 5
100  PRINT X$(J,I);" ";
110  NEXT J
120  PRINT
130  NEXT I
200  DATA A,B,C,D,E,F,G,H,I,J,K,L,M,N,O
```

CHAPTER FIVE

1.

```
10  INPUT "NAME? ";N$
20  PRINT N$
30  IF N$ = "DONALD" THEN  GOSUB 100
40  GOTO 10
50  END
100  PRINT "------"
110  RETURN
```

2.

```
1  REM  C = Card number
2  REM  F = Face value
3  REM  S = Suit
4  REM
10  FOR C = 1 TO 5
20 S =  INT (4 *  RND (1) + 1)
30 F =  INT (13 *  RND (1) + 1)
40  PRINT "CARD ";C;" IS THE ";F;" OF ";
50  ON S GOTO 100,200,300,400
60  NEXT C
70  END
100  PRINT "HEARTS"
110  GOTO 60
200  PRINT "SPADES"
210  GOTO 60
300  PRINT "DIAMOMDS"
310  GOTO 60
400  PRINT "CLUBS"
410  GOTO 60
```

3.

```
10  PRINT "ENTER A (BETWEEN 0 AND 3) ";
20  GET A: PRINT A
30  IF A > 3 OR A < 0 THEN 10
40  ON A + 1 GOSUB 100,150,200,250
50  END
100  PRINT "YOU ARE A ZERO"
110  RETURN
150  PRINT "DANIELLE LOVES MICHAEL"
160  RETURN
200  PRINT "MICHAEL LOVES DANIELLE"
210  RETURN
250  PRINT "WHO IS DANIELLE?"
260  RETURN
```

4a. The FOR-TO-NEXT loop in line 20 is missing the STEP command. Line 20 should read:

20 FOR X = 10 TO 1 STEP −1

4b. The product in line 30 is always positive. The best way to correct this error in logic would be to eliminate line 40 and replace it with:

40 PRINT "THE PRODUCT IS POSITIVE"

CHAPTER SIX

1.

```
10  HOME : GR : COLOR= 4
20  HLIN 0,39 AT 0: HLIN 0,39 AT 39
30  VLIN 0,39 AT 0: VLIN 0,39 AT 39
```

2.

```
10  HOME : GR : COLOR= 1
20  FOR X = 0 TO 39
30  VLIN 0,39 AT X
40  NEXT X
50  COLOR= 2
60  PLOT 19,19
```

3.

```
10  HOME : GR
20  FOR X = 1 TO 15
30  COLOR= X: VLIN 0,39 AT 2 * X + 4
40  NEXT X
```

4.

```
10  HGR
20  INPUT "X,Y? ";X,Y
30  IF X < 0 OR X > 279 THEN  PRINT "OUT OF RANGE! ": GOTO 20
40  IF Y < 0 OR Y > 159 THEN  PRINT "OUT OF RANGE! ": GOTO 20
50  HCOLOR= 8 *  RND (1)
60  HPLOT X,Y
70  GOTO 20
```

5.

```
10  HGR
20  FOR I = 1 TO 7
30  HCOLOR= I
40  HPLOT 0,11 * (I - 1) TO 279,11 * (I - 1)
50  NEXT I
```

6.

```
1  REM  X0,Y0 = coordinate of origin
2  REM  X = value to be used in the equation
3  REM  Y = result of the equation
4  REM  X1,Y1 = coordinates to be plotted on HI-RES screen
5  REM
10  HGR2 : HCOLOR= 7
20 X0 = 135:Y0 = 100
30  REM  draw axes
40  HPLOT 0,Y0 TO 279,Y0
50  HPLOT X0,0 TO X0,191
60  REM
100  REM  graph function
110  FOR X1 = 0 TO 279
120 X = X1 - X0
130 Y = .2 * X ^ 2
140 Y1 = Y0 - Y
150  IF (Y1 >  = 0) AND (Y1 <  = 191) THEN  HPLOT X1,Y1
160  NEXT X1
```

CHAPTER SEVEN

1. ④

2.

```
10  INPUT "FIRST COORDINATE (X1,Y1)? ";X1,Y1
20  INPUT "SECOND COORDINATE (X2,Y2)? ";X2,Y2
30  PRINT "DISTANCE IS ";
40 D =  SQR ((X1 - X2) ^ 2 + (Y1 - Y2) ^ 2)
50  PRINT D
```

3.

```
10  DEF  FN DR(X) = X * 3.141592654 / 180
20  DEF  FN RD(X) = X * 180 / 3.141592654
30  INPUT "DEGREES? ";D
40  PRINT "THAT IS "; FN DR(D);" RADIANS."
50  PRINT
60  INPUT "RADIANS? ";R
70  PRINT "THAT IS "; FN RD(R);" DEGREES."
```

CHAPTER EIGHT

1. (A) 16 BITS FOR THE INTEGER 6167

(B) 72 BITS - 8 FOR EACH LETTER IN 'APRIL' PLUS 16 EACH FOR THE INTEGERS 1 AND 1963

(C) 32 BITS - 42864 IS AN INTEGER, BUT IT IS OUTSIDE THE ALLOWABLE RANGE FOR INTEGER STORAGE, SO IT MUST BE STORED AS A FLOATING POINT NUMBER.

(D) 32 BITS FOR THE FLOATING POINT NUMBER 35.78

2.

```
10  INPUT "INPUT A LETTER? ";L$
20  PRINT "THE ASCII OF '";L$;"' IS "; ASC (L$)
30  PRINT "TWO LETTERS AFTER '";L$;"' IS ";
40 C =  ASC (L$):C = C + 2: IF C > 90 THEN C = C - 26
50  PRINT "'"; CHR$ (C);"'"
```

CHAPTER NINE

1.

```
1  REM  M$ = month name
2  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN MONTHS OF THE YEAR"
30  PRINT DOS$;"WRITE MONTHS OF THE YEAR"
40  FOR I = 1 TO 12
50  READ M$
60  PRINT M$
70  NEXT I
80  PRINT DOS$;"CLOSE MONTHS OF THE YEAR"
90  REM
100  DATA  JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY
110  DATA  AUGUST,SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER
```

2.

```
1  REM  M$ = month name
2  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT "ODD MONTHS OF THE YEAR -"
30  PRINT DOS$;"OPEN MONTHS OF THE YEAR"
40  PRINT DOS$;"READ MONTHS OF THE YEAR"
50  FOR I = 1 TO 12
60  INPUT M$
70  IF I / 2 =  INT (I / 2) THEN 90
80  PRINT M$
90  NEXT I
100  PRINT DOS$;"CLOSE MONTHS OF THE YEAR"
```

3.

```
1  REM  M$ = month name
2  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN MONTHS OF THE YEAR"
30  PRINT DOS$;"OPEN TEMP"
40  FOR I = 1 TO 12
50  PRINT DOS$;"READ MONTHS OF THE YEAR"
60  INPUT M$
70  IF I / 2 =  INT (I / 2) THEN 100
80  PRINT DOS$;"WRITE TEMP"
90  PRINT M$
100  NEXT I
110  PRINT DOS$;"CLOSE TEMP"
120  PRINT DOS$;"CLOSE MONTHS OF THE YEAR"
130  PRINT DOS$;"DELETE MONTHS OF THE YEAR"
140  PRINT DOS$;"RENAME TEMP,MONTHS OF THE YEAR"
150  PRINT "The even months of the year have been removed"
```

4.

```
1  REM  M$ = month name
2  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN MONTHS OF THE YEAR,L10"
30  FOR I = 1 TO 12
40  PRINT DOS$;"WRITE MONTHS OF THE YEAR,R";I
50  READ M$
60  PRINT M$
70  NEXT I
80  PRINT DOS$;"CLOSE MONTHS OF THE YEAR"
90  REM
100  DATA  JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY
110  DATA  AUGUST,SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER
```

5.

```
1  REM  M$ = month name
2  REM
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT "ODD MONTHS OF THE YEAR -"
30  PRINT DOS$;"OPEN MONTHS OF THE YEAR,L10"
40  FOR I = 1 TO 12 STEP 2
50  PRINT DOS$;"READ MONTHS OF THE YEAR,R";I
60  INPUT M$
80  PRINT M$
90  NEXT I
100  PRINT DOS$;"CLOSE MONTHS OF THE YEAR"
```

Answers

To Odd Numbered Exercises

CHAPTER ONE

1.
```
10  PRINT "A"
20  PRINT " B"
30  PRINT "  C"
40  PRINT "ABCD"
```

3.
```
]RUN
The value of B
19

]
```

5.
```
10  READ A,B
20  PRINT "The sum is ";A + B
30  GOTO 10
40  DATA  12,8,9,5
```

7.
```
10  INPUT "Price, Number of loaves? ";P,N
20  PRINT "Total spent = $";P * N / 100
```

9.
```
10  INPUT X,Y
20  PRINT "X = ";X,"Y = ";Y,"X*Y = ";X * Y
30  GOTO 10
```

11.

```
]RUN
300                 510
.3   .51

]
```

13.

```
]PRINT "AAA";111,222;"AAA","333";"    ";16-3*2
AAA111          222AAA          333    10

]
```

15.

```
]RUN
abcdxyz
abcd7
7xyz
-4xyz

]
```

17.

```
10  PRINT "Total mileage for the week is: ";
20  PRINT 2 + 3 + 4 + 3 + 5
```

19.

```
10  READ X,Y
20 A = 12 * X + 7 * Y
30  PRINT A
40  GOTO 10
50  DATA  3,2,7,9,12,-4
```

21.

```
10  INPUT "Length, Width, Height? ";L,W,H
20  PRINT "Volume is ";L * W * H
```

23.

```
10  INPUT "Height? ";H
20  INPUT "Width? ";W
30  INPUT "Length? ";L
40  INPUT "Presence? ";P
50  PRINT "Your object uses ";H * W * L * P;" Tesseracts
           in four space."
```

25.

```
10  INPUT "How many books have you borrowed? ";B
20  INPUT "How many days late are they? ";L
30  PRINT "You owe $ ";.10 * B * L
```

27.

```
10  INPUT "What is the player's name? ";P$
20  PRINT "What is ";P$;"'s";
30  INPUT " wage ";W
40 T = W * .44
50  PRINT P$;" would keep $";W - T
60  PRINT P$;" would pay $";T;" in taxes."
```

29.

```
10  INPUT "What is the base? ";B
20  INPUT "What is the altitude? ";A
30  PRINT
40  PRINT "The area is ";A * B / 2
```

31.

```
10  READ S1,G1,S2,G2,S3,G3,S4,G4,S5,G5
20 S = S1 + S2 + S3 + S4 + S5
30 G = G1 + G2 + G3 + G4 + G5
40 T = S + G
50  PRINT "Sloth's total vote was ";S
60  PRINT "His total percentage was ";100 * S / T
70  PRINT
80  PRINT "Graft's total vote was ";G
90  PRINT "His total percentage was ";100 * G / T
100  DATA  522,210,313,721,1003,822,413,1107,516,1700
```

33.

```
10  INPUT "Month, day, and year of birth? ";M1,D1,Y1
20 D9 = Y1 * 365 + M1 * 30 + D1
30  INPUT "Today's month, day, and year? ";M2,D2,Y2
40 D8 = Y2 * 365 + M2 * 30 + D2
50 S = 8 * (D8 - D9)
60  PRINT "You have slept about ";S;" hours."
```

CHAPTER TWO

1.

```
10  INPUT A,B
20  IF A < B THEN  PRINT A;" is less than ";B
30  IF A > B THEN  PRINT A;" is greater than ";B
40  IF A = B THEN  PRINT A;" is equal to ";B
50  GOTO 10
```

3a.

```
]PRINT 3^2^3
729.000003
```

b.

```
]PRINT 5-4^2
-11
```

c.

```
]PRINT 3*(5+16)
63
```

d.

```
]PRINT 5+3*6/2
14
```

e.

```
]PRINT 640/10/2*5
160
```

f.

```
]PRINT 5+3*4-1
16
```

g.
```
]PRINT 2^3^2
64
```

h.
```
]PRINT 2^(3^2)
512.000002
```

i.
```
]PRINT 64/4*.5+((1+5)*2^3)*1/(2*4)
14
```

5.
```
10   INPUT A$,B$: PRINT A$,B$: PRINT B$,A$
```

7.
```
10   INPUT A$
20   IF A$ > "DOWN" AND A$ < "UP" THEN  PRINT "A$ is between"
```

9.
```
10   INPUT X
20   IF X <  = 25 OR X >  = 75 THEN  PRINT "Not in the interval"
     : GOTO 10
30   PRINT "in the interval": GOTO 10
```

11.
```
10   FOR I = 11 TO  - 11 STEP  - 2
20   PRINT I ^ 3
30   NEXT I
```

13.
```
10   FOR I = 1 TO 20
20   PRINT "*";
30   NEXT I
```

15.
```
10   FOR I = 10 TO 97 STEP 3
20   PRINT I;" ";
30   NEXT I
```

17.
```
10   INPUT "Creature? ";X$
20   RESTORE
30   FOR I = 1 TO 6
40   READ C$,W$
50   IF C$ = X$ THEN 90
60   NEXT I
70   PRINT "Creature ";X$;" not found."
80   GOTO 10
90   PRINT "You can kill a ";C$;" with a ";W$
100   GOTO 10
110   DATA  LICH,FIRE BALL, MUMMY,FLAMING TORCH,WEREWOLF,
            ILVER BULLET
120   DATA  VAMPIRE,WOODEN STAKE,MEDUSA,SHARP SWORD,
            TRIFFID,FIRE HOSE
```

19.
```
#10 #20 #30 #40 #50 #60 #10 #20 #30 #40
#50 #60 #10 #20 #30 #40 #50
```

21.

```
10  FOR I = 1 TO 10
20  PRINT : PRINT "----------------------"
30  PRINT "  Happy Holiday Motel"
40  PRINT "        Room ";I
50  PRINT "----------------------"
60  NEXT I
```

23.

```
10  PRINT "X","X^2","X^3"
20  PRINT
30  FOR I = 2 TO 10 STEP 2
40  PRINT I,I ^ 2,I ^ 3
50  NEXT I
```

25.

```
10  INPUT N
20  FOR H = 1 TO N: REM  H is the height of the triangle
30  IF W = H THEN 70: REM  W is the width
40  PRINT "*";
50 W = W + 1
60  GOTO 30
70  PRINT :W = 0: REM  Reset width for each H
80  NEXT H
```

27.

```
10  FOR I = 1 TO 5
20  READ N$,P: REM  Get name, performance
30  IF P >  = 75 THEN 120
40  PRINT : PRINT "Dear ";N$;","
50  PRINT " I am sorry that I must fire you."
60  PRINT "You have been such a fine employee"
70  PRINT "with a performance rating of ";P;"%"
80  PRINT "I'm sure you'll have no trouble"
90  PRINT "finding another job."
100  PRINT  TAB( 20);"Sincerely,"
110  PRINT : PRINT  TAB( 20);"George Shwabb": PRINT
120  NEXT I
130  DATA  Oakley,69,Howe,92,Anderson,96,Olley,88,Goerz,74
```

29.

```
10  INPUT "Hours worked? ";H
20  INPUT "Hourly wage? ";W
30 M = W * H
40  IF H > 40 THEN M = M + W / 2 * (H - 40)
50  PRINT "The wage for the week is $";M
```

31.

```
1  REM  A = Amount of money left
10 A = 200: REM  Initial amount
20  INPUT "How much does the item cost? ";C
30  IF C = 0 THEN  END
40 A = A - 1.05 * C: REM  1.05*C is cost with tax
50  IF A < 0 THEN 90
60  PRINT "Your total is now $";A
70  PRINT
80  GOTO 20
90  PRINT "You don't have enough money."
100 A = A + 1.05 * C
110  GOTO 20
```

CHAPTER THREE

1.

```
10  FOR I = 1 TO 10
20 R =  RND (1)
30  IF R > .5 THEN  PRINT R
40  NEXT I
```

3.

```
10  INPUT N
20  IF N =  INT (N) THEN  PRINT N
30  GOTO 10
```

5.

```
10 N =  INT (4 *  RND (1) + 2)
20 D =  INT (4 *  RND (1) + 1)
30 Q =  INT (4 *  RND (1))
40 F = .05 * N + .1 * D + .25 * Q: REM  total amount found
50  PRINT "You found $";F
60  IF F > .99 THEN  PRINT "You can buy lunch": END
70  PRINT "Sorry, you can't buy lunch"
```

7.

```
10  REM   A = Amount in bank
20  REM  I = Week number
30 A = 11: REM  Initial Amount
40  FOR I = 1 TO 4
50  PRINT "Week ";I;", How many pennies do you have";
60  INPUT N
70 A = A + N
80  PRINT "Your total is now $";A / 100
90  NEXT I
```

9.

```
10  HOME
20  FOR X = 1 TO 24
30  HTAB (X): VTAB (X)
40  PRINT "*"
50  NEXT X
```

11.

```
]RUN
1
 2
  3
   4
                 1
                 2
                 3
                 4

]
```

11b.

```
]RUN
123.4               123.5
123.45              123.46

]
```

13.

X	Y
$\boxed{1}$	$\boxed{5}$
2	0
2	2
2	1
3	1
3	-1
$\boxed{3}$	$\boxed{2}$

15.

```
10  FOR I = 13 TO 147 STEP 2
20 S = S + I
30  NEXT I
40  PRINT "The sum = ";S
```

17.

```
10  FOR I = 1 TO 1000
20 R =  INT (9 *  RND (1) + 1)
30  IF R / 2 =  INT (R / 2) THEN E = E + 1
40  NEXT I
50  PRINT "There were ";1000 - E;" odd integers."
60  PRINT "There were ";E;" even integers."
```

19.

```
10  PRINT "Date","Balance"
20 B = 1000:Y = 1984
30 B = B * 1.05:Y = Y + 1
40  PRINT "Jan 1, ";Y,"$"; INT (100 * B + .5) / 100
50  IF B > 2000 THEN  END
60  GOTO 30
```

21.

```
10  READ V
20 PI = 3.14159265
30 R = (.75 * V / PI) ^ (1 /.3)
40  PRINT "A sphere with volume ";V;" cm.^3"
50  PRINT "has a radius of about "; INT (100 * R + .5) /
    100;" cm.
     "
60  PRINT
70  GOTO 10
100  DATA  690,720,460,620
```

23.

```
10  INPUT "Your string? ";A$
20  PRINT A$;
30 P =  POS (0)
40  PRINT
50  PRINT "The length of ";A$;" is ";P
```

25.

```
10  FOR I = 1 TO 20
20 N =  INT (100 *  RND (1) + 1)
30  IF N = 1 THEN 90
40  IF N = 2 THEN 80
50  FOR X = 2 TO  SQR (N)
60  IF N / X =  INT (N / X) THEN 90
70  NEXT X
80  PRINT N;" ";
90  NEXT I
```

27.

```
10 Q =  INT (8 *  RND (1))
20 D =  INT (5 *  RND (1))
30 P =  INT (10 *  RND (1))
40 V = .25 * Q + .1 * D + .01 * P
50 T = Q + D + P
60  PRINT "There are ";T;" coins, totalling $";V
70  PRINT "Guess how many quarters, dimes, and pennies I have."
80  FOR I = 1 TO 3
90  INPUT "Guess? ";Q1,D1,P1
100  IF Q1 = Q AND D1 = D AND P1 = P THEN  PRINT "You got it!":
     END

110  PRINT "Incorrect."
120  NEXT I
130  PRINT "Hah, you didn't get it."
140  PRINT "I had ";Q;" quarters, ";D;" dimes, and ";P;" pennies."
```

29.

```
5  HOME
10 R = 5:X = 20:Y = 12
20  FOR Y1 =  - 5 TO 5
30 X1 =  SQR (R ^ 2 - Y1 ^ 2)
40  HTAB ( INT (X + X1 + .5)): VTAB ( INT (Y + Y1 + .5))
50  PRINT "*";
60  HTAB ( INT (X - X1 + .5)): VTAB ( INT (Y + Y1 + .5))
70  PRINT "*";
90  NEXT Y1
```

31.

```
10 X1 =  INT (40 *  RND (1) + 1):Y1 =  INT (24 *  RND (1) + 1)
20 X2 =  INT (40 *  RND (1) + 1):Y2 =  INT (24 *  RND (1) + 1)
30  HOME : HTAB (X1): VTAB (Y1): PRINT "*";
40  HTAB (X2): VTAB (Y2): PRINT "*";
50  FOR Z = 1 TO 10
60  HTAB (1): VTAB (22)
70  INPUT "Coordinates (X,Y)? ";X,Y
75  HTAB (X): VTAB (Y): PRINT "X"
80  IF X = X1 AND Y = Y1 THEN T = T + 1: GOTO 130
90  IF X = X2 AND Y = Y2 THEN T = T + 1: GOTO 150
95  HTAB (1): VTAB (22)
100  PRINT "MISSED "
```

```
110  IF T = 2 THEN 170
120  NEXT Z: GOTO 190
130  PRINT "GOT IT!": HTAB (X1): VTAB (Y1)
140  PRINT " ";: GOTO 110
150  PRINT "GOT IT!": HTAB (X2): VTAB (Y2)
160  PRINT " ";: GOTO 110
170  FLASH : PRINT "YOU HAVE DEFEATED THE EMPIRE!": NORMAL
180  END
190  HOME : VTAB (10): HTAB (5)
200  PRINT "The empire is victorious you ";
210  FLASH : PRINT "TURKEY": NORMAL
```

CHAPTER FOUR

1.

```
10  FOR I = 1 TO 8
20  FOR I1 = 1 TO 30
30  PRINT "*";
40  NEXT I1
50  PRINT
60  NEXT I
```

3.

```
10  FOR I = 1 TO 6
20  INPUT "Enter X(I) ";X(I)
30  NEXT I
40  FOR I = 1 TO 5 STEP 2
50  PRINT I,X(I)
60  NEXT I
70  FOR I = 2 TO 6 STEP 2
80  PRINT I,X(I)
90  NEXT I
```

5.

```
10  DIM A(4,12)
20  FOR I = 1 TO 4
30  FOR J = 1 TO 12
40 A(I,J) = 3 * I + J * J
50  NEXT J
60  NEXT I
70  INPUT "N ";N
80  FOR I = 1 TO 12
90  PRINT A(N,I);" ";
100  NEXT I
110  PRINT
120  GOTO 70
```

7a.

```
]RUN
1               5
1               6
2               5
2               6
3               5
3               6

]
```

7b.

```
]RUN
10
10
10
12
12
11
14
13
11

]
```

7c.

```
]RUN
45              89              35

]
```

9.

```
10   FOR X = 40 TO 1 STEP  - 1
20   FOR Y = 1 TO 10
30   READ N
40   IF N = X THEN  PRINT N;" ";
50   NEXT Y
60   RESTORE
70   NEXT X
80   DATA  5,27,37,16,27,8,2,40,1,9
```

11.

```
10   DIM N(100)
20   FOR X = 1 TO 100
25   REM   Get a random number
30 N(X) =  INT (99 *  RND (1) + 1)
35   REM   Loop through all previous numbers
40   FOR Y = 0 TO X - 1
45   REM   Check for any duplicates
50   IF N(X) = N(Y) THEN 90
60   NEXT Y
70   NEXT X
80   END
90   PRINT "Duplicate after ";X;" numbers"
100  FOR I = 1 TO X
110  PRINT N(I);" ";
120  NEXT I
```

13.

```
10  FOR I = 3 TO 50
20  FOR J = I + 1 TO 50
30  FOR K = J + 1 TO 50
40  IF K * K > I * I + J * J THEN 70
50  IF K * K = I * I + J * J THEN  PRINT I,J,K
60  NEXT K
70  NEXT J
80  NEXT I
```

15.

```
10  DIM N(20)
20  FOR I = 1 TO 20
30 N(I) =  INT (90 *  RND (1) + 10)
40  NEXT I
50  PRINT "Odd integers: ";
60  FOR I = 1 TO 20
70  IF N(I) / 2 <  >  INT (N(I) / 2) THEN  PRINT N(I);" ";
80  NEXT I
90  PRINT : PRINT "Even integers: ";
100  FOR I = 1 TO 20
110  IF N(I) / 2 =  INT (N(I) / 2) THEN  PRINT N(I);" ";
120  NEXT I
```

17a.

```
10  DIM N$(5,6)
20  INPUT "What day and time would you like? ";D,T
30  IF N$(D,T) <  > "" THEN  PRINT "That time is taken.": GOTO 20
40  INPUT "What is your name? ";N$(D,T)
50  PRINT "Thank you very much."
60  GOTO 20
```

17b.

```
10  DIM N$(5,6)
15  INPUT "Are you the doctor? ";A$
17  IF A$ = "YES" THEN 70
20  INPUT "What day and time would you like? ";D,T
30  IF N$(D,T) <  > "" THEN  PRINT "That time is taken.": GOTO 20
40  INPUT "What is your name? ";N$(D,T)
50  PRINT "Thank you very much."
60  GOTO 15
70  INPUT "Which day? ";D
80  FOR T = 1 TO 6
90  IF N$(D,T) = "" THEN  PRINT "Sanka break": GOTO 110
100  PRINT N$(D,T)
110  NEXT T
120  GOTO 15
```

19.

```
10  DIM N(100)
20 R =  INT (100 *  RND (1) + 1)
30  FOR X = 1 TO 100
40  INPUT "Guess? ";G
50  IF G = R THEN 150
60  FOR Z = 1 TO X - 1
70  IF N(Z) = G THEN 130
80  NEXT Z
90 N(X) = G
100  IF G > R THEN  PRINT "Lower": GOTO 120
110  PRINT "Higher"
120  NEXT X
130  PRINT "WAKE UP! You guessed that number before."
140  GOTO 40
150  PRINT "Correct."
```

21.

```
1  REM  P() = Point values for the board
2  REM  B() = P, with a zero entered when a penny hits the board
3  REM
4  REM
10  DIM P(6,6),B(6,6)
20  REM  Set up outside of board for 1 point
30  FOR I = 1 TO 6
40 P(1,I) = 1:P(6,I) = 1:P(I,6) = 1:P(I,1) = 1
50  NEXT I
60  REM  Set up board positions worth 2 points
70  FOR I = 2 TO 5
80 P(2,I) = 2:P(5,I) = 2:P(I,2) = 2:P(I,5) = 2
90  NEXT I
100  REM  Set up the 3 point positions
110 P(3,3) = 3:P(3,4) = 3:P(4,3) = 3:P(4,4) = 3
120  REM  set array B equal to array P
130  FOR Q = 1 TO 6
140  FOR W = 1 TO 6
150 B(Q,W) = P(Q,W)
160  NEXT W
170  NEXT Q
180  REM  get 10 random rows and columns for penny tosses
190  FOR T = 1 TO 10
200 R =  INT (6 *  RND (1) + 1):C =  INT (6 *  RND (1) + 1)
210 B(R,C) = 0: REM  Set hit positions to 0
220 S = S + P(R,C): REM  sum of point values
230  NEXT T
240  REM  Print out resultant board'
250  FOR R = 1 TO 6
260  FOR C = 1 TO 6
270  IF B(R,C) = 0 THEN  PRINT "X ";: GOTO 290
280  PRINT B(R,C);" ";
290  NEXT C
300  PRINT
310  NEXT R
320  PRINT : PRINT "Score = ";S
```

23.

```
1  REM  A$() = Letters of the alphabet
2  REM  W$() = The word formed
10  DIM A$(26)
20  FOR I = 1 TO 26
30  READ A$(I)
40  NEXT I
50  FOR W = 1 TO 15
60 R =  INT (7 *  RND (1) + 1): REM  Random length
70  FOR L = 1 TO R: REM  Get word of length R
80 W$ = W$ + A$( INT (26 *  RND (1) + 1))
90  NEXT L
100  PRINT W$
110 W$ = "": REM  Blank W$ for next word
120  NEXT W
200  REM
210  DATA  A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
```

25.

```
10 M = 500
20  FOR X = 1 TO 21
30  FOR Q = 1 TO 4
40 M = M + M * .06 / 4
50  NEXT Q
60  PRINT "At the end of the year ";X;" there is $";M
70 M = M + 60
80  NEXT X
```

CHAPTER FIVE

1.

```
10  GOSUB 200
20  FOR J = 1 TO 40
30  PRINT "*";
40  NEXT J
50  GOSUB 200
60  FOR J = 1 TO 3
70  PRINT "!",
80  NEXT J
90  GOSUB 200
100  FOR J = 1 TO 20
110  PRINT "AB";
120  NEXT J
130  END
200  PRINT
210 I = I + 1
220  PRINT "Part ";I
230  RETURN
```

3.

```
5  PRINT "Pick 1, 2, or 3"
10  GET A
20  IF A < 1 OR A > 3 THEN 10
30  ON A GOTO 40,60,80
40  PRINT "Fame is yours if no one eats your Apple."
50  END
60  PRINT "Blame is yours if someone beats your Apple."
70  END
80  PRINT "Success is yours if you don't mistreat your Apple."
90  END
```

5a.

```
#10 #20 #30 #70 #40
#10 #20 #30 #50 #40
#10 #20 #30 #60 #40
#10 #20 #90 #100
```

5b.

```
]RUN
N=3                Z=1                P=4

]
```

7.

```
10  READ A,B,C
20  GOSUB 200
30  IF L = 1 THEN  PRINT "Not a right triangle": GOTO 10
40  PRINT "Area=";A * B / 2,"Perimeter=";A + B + C
50  GOTO 10
200 L = 0: REM  Reset the "flag"
210  IF C >  = A + B THEN 250
220  IF B >  = A + C THEN 250
230  IF A >  = B + C THEN 250
240  IF A * A + B * B = C * C THEN 260
250 L = 1
255  REM  Note that A^2 produces a rounding error in some cases.
256  REM  When you compare squared values, it is safer to use A*A.
260  RETURN
270  DATA  3,4,5,0,1,1,2,2,2,12,5,13
280  END
```

9.

```
10  PRINT "WITHDRAWAL(1),DEPOSIT(2),"
20  PRINT "CALCULATE INTEREST(3), OR EXIT(4)
30  PRINT "1,2,3, OR 4? ";: GET D
40  IF D < 1 OR D > 4 THEN 30
45  PRINT D
50  ON D GOSUB 100,200,300,400
60  PRINT "Your balance stands at $"; INT (100 * B + .5) / 100
70  PRINT : GOTO 10
100  REM   Withdrawal routine
110  INPUT "How much would you like to withdraw? ";A
120  IF A >  = 0 AND B - A >  = 0 THEN B = B - A: RETURN
130  GOTO 110
200  REM   Deposit routine
```

```
210  INPUT "How much would you like to deposit? ";A
220  IF A >  = 0 THEN B = B + A: RETURN
230  GOTO 210
240  RETURN
300  INPUT "How many months since last calculation? ";M
310 Q = M / 3
320  IF Q < 1 THEN  PRINT "Too soon": RETURN
330  FOR C = 1 TO Q
340 B = B + .0575 / 4 * B
350  NEXT C
360  RETURN
400  END
```

11.

```
Label1: Deal Player two cards
        Deal Dealer two cards
        Reveal both of Player's cards
        Reveal one of Dealer's cards

Label2: Does the Player want more cards?
        If not, goto Label3
        Deal Player another card
        If Player's TOTAL >21 then Dealer wins : goto Label5
        Goto Label2

Label3: REM Dealer takes cards routine
        If Dealer's TOTAL >16 and Dealer's TOTAL <21
          then goto Label14
        Deal Dealer another card
        If Dealer's TOTAL >21 then Player wins : goto Label5
        Goto Label3

Label4: REM Compare card totals
        If Dealer's TOTAL >= Player's TOTAL then
          Dealer wins : goto Label5
        Player wins

Label5: Does the Player want to play again?
        If the Player wants to play again then goto Label1
        END
```

13. (a) This program will generate an "?OUT OF DATA ERROR IN 10". Line 10 attempts to read a fourth data element from line 40. A possible correction might read:

```
40 DATA 2,3,4,5
```

where the value 5 becomes the fourth data element.

(b) This program will endlessly print .5 because no new data is read in at line 10. Correct line 40 as follows:

```
40 GOTO 10
```

(c) Lines 20 and 30 each contain syntax errors. The function "* /" referenced at line 20 is illegal. The statement may be corrected to read:

```
20 PRINT A*B+C
```

to indicate multiplication, or:

```
20 PRINT A/B+C
```

to indicate division of A by B. Line 30 may be corrected by placing quotation marks around "D/F=" to read:

```
30 PRINT “D/F=”;D/F
```

In addition, the variables D and F should be defined, or else a division by zero will result in line 30.

(d) The conditional clause at line 20 is incomplete. The user has not specified what variable should be less than 10. The obvious variable is F, so that line 20 reads:

```
20 IF F>5 OR F<10 THEN 40
```

A comma or a semicolon must be inserted between the variables F and G at line 40 so that the computer understands that they are two separate elements:

```
40 PRINT F,G
```

(e) The FOR . . . TO . . . NEXT loops are improperly nested:

```
10  FOR X=1 TO 8
20    FOR Y=1 TO 3
30      X=X+Y
40        NEXT X
50  NEXT Y
60  PRINT X
70  END
```

Line 40 and 50 may be corrected as follows:

```
40  NEXT Y
50  NEXT X
```

Also, changing the value of variables used in FOR . . . TO . . . NEXT loops is not recommended practice such as at line 30.

(f) This program will continuously print the sum of 3, 6 and 9 since new data is not being read at line 10. Correct line 40 as follows:

```
40  GOTO 10
```

15.
```
]RUN
-6                          6

]
```

17. The computer will return the error "?SYNTAX ERROR IN 20". Since it appears that the variable "X" is intended to contain formula weight, line 20 may be corrected to read:

```
20 INPUT ''FORMULA WEIGHT'';X
```

(b) When this program attempts to branch to line 50 at line 35, it will not find lin 50, and the error message "?UNDEF'D STATEMENT ERROR IN 35" will be printed. Change line 35 to:

```
35 IF Y=X THEN A=A−1 : GOTO 55
```

(c) Note that both lines 10 and 20 do not contain the same number of closing parentheses as opening parentheses, thereby causing an "?ILLEGAL QUANTITY ERROR IN 10". The computer reports the first error encountered so the errors in line 20 and 30 go undiscovered. The first opening parenthesis is not necessary in either line 10 or 20. Line 30 should read:

```
IF X1>X2 THEN 50
```

(d) Upon encountering line 30, the computer will print the error message "?SYNTAX ERROR IN 30". The line in question should read:

```
30 IF X>200 THEN 10
```

(e) When the computer runs this program, it will print "?TYPE MISMATCH IN 20" and terminate the run. The problem arises from trying to add a floating point variable to a string variable. Variables can be of several types. They can be integers (which means they have no decimal places), floating point or real (which means they have decimal places), or string variables. Assuming the intent of the program was to sum all the numbers between 1 and 26, the best way to correct this program is to change all occurrences of "A$" to "A".

```
20      A=A+X
40   PRINT A
```

19. The computer stores all decimal numbers in binary form. Because of this, small "rounding errors" are introduced into stored numbers. When these errors are compounded as in the large summation at lin 30 (100 and 1000 times), it leads to very visible miscalculations as seen when this program is run. In this case the error was small.

21.
```
10  ONERR  GOTO 100
20  FOR X =  - 5 TO 5
30 J = 10 / X
40  PRINT J
50  NEXT X
60  GOTO 130
100 N =  PEEK (222):L =  PEEK (218) + 256 *  PEEK (219)
110  IF N = 133 AND L = 30 THEN  PRINT "Division by zero":
     X = 1: RESUME
120  PRINT "Error #";N;" at line ";L
130  END
```

CHAPTER SIX

1.
```
10  GR : COLOR= 9
20  HLIN 19,21 AT 10
30  HLIN 19,21 AT 11
```

3.
```
10  GR : COLOR= 14
20  PLOT 15,18
30  VLIN 20,25 AT 15
```

5.
```
10  HGR : HCOLOR= 6
20  HPLOT 250,20 TO 250,110 TO 130,110 TO 250,20
```

7.
```
10  HGR : HCOLOR= 6
20  HPLOT 10,10 TO 130,110 TO 130,10
30  HPLOT  TO 70,10 TO 70,110
40  HPLOT  TO 10,110 TO 10,10
```

9.
```
1  REM  PR = pizza production
2  REM  C = column number
3  REM  PR() = total production for each employee
4  REM
10  REM  read in pizza production data
20  FOR I = 1 TO 4
30  FOR J = 1 TO 7
40  READ PR
50 P(I) = P(I) + PR
60  NEXT J
65 P(I) = P(I) / 7: REM  take average
70  NEXT I
100  REM  draw bar graph
110  HOME : GR : COLOR= 15
120  FOR N = 1 TO 4
130  FOR C = 1 TO 3
140  VLIN 40 - P(N),39 AT (N - 1) * 10 + C
```

```
150  NEXT C
160  NEXT N
170  REM
200  REM  label each bar
210  PRINT
220  FOR I = 1 TO 4
230  READ N$: PRINT  TAB( 10 * (I - 1) + 1);N$;
240  NEXT I
250  REM
290  REM  pizza production data
300  DATA  18,12,9,10,16,22,14
310  DATA  12,21,19,16,28,20,22
320  DATA  18,20,14,19,11,16,23
330  DATA  23,27,18,16,21,14,24
340  REM
390  REM  name data
400  DATA  PAPA,DELLACONA,RICARDO,FERRARA
```

11a.

```
10  GR
20  FOR X = 0 TO 19
30  FOR Y = 0 TO 39
40  COLOR= 16 *  RND (1)
50  PLOT X,Y
60  NEXT Y
70  NEXT X
80  REM
90  REM  duplicate left portion
100  FOR X = 20 TO 39
110  FOR Y = 0 TO 39
120  COLOR=  SCRN( X - 20,Y)
130  PLOT X,Y
140  NEXT Y
150  NEXT X
```

11b.

```
10  GR
20  FOR X = 0 TO 19
30  FOR Y = 0 TO 39
40  COLOR= 16 *  RND (1)
50  PLOT X,Y
60  NEXT Y
70  NEXT X
80  REM
90  REM  duplicate left portion
100  FOR X = 39 TO 20 STEP  - 1
110  FOR Y = 0 TO 39
120  COLOR=  SCRN( 39 - X,Y)
130  PLOT X,Y
140  NEXT Y
150  NEXT X
```

13.

```
10  HOME : GR : COLOR= 3
20  DIM C(21,2)
30  FOR I = 1 TO 7
40  READ L(I): REM  location of start of data for roll "I" in C()
50  NEXT I
60  REM  read coordinates of "dots"
70  FOR I = 1 TO 21
80  READ C(I,1): REM  X coordinate of dot
90  READ C(I,2): REM  Y coordinate
100  NEXT I
110  REM
200 R =  INT (6 *  RND (1) + 1): REM  random roll
210  GOSUB 300
220  GET N$
230  GR : COLOR= 3: GOTO 200
240  REM
300  REM  subroutine to draw die
310  FOR I = L(R) TO L(R + 1) - 1
320 X = C(I,1)
330 Y = C(I,2)
340  PLOT X,Y
350  NEXT I
360  RETURN
370  REM
400  REM  index data
410  DATA  1,2,4,7,11,16,22
420  REM
500  REM  coordinate data
510  DATA  20,20,0,0,39,39
520  DATA  0,0,20,20,39,39
530  DATA  0,0,39,0,0,39,39,39
540  DATA  0,0,39,0,20,20,39,39,0,39
550  DATA  0,0,39,0,0,20,39,20,0,39,39,39
```

15.

```
10  INPUT "A,B,R? ";A,B,R
20  HGR : HCOLOR= 3
30  FOR X = A - R TO A + R
40 P =  SQR (R ^ 2 - (X - A) ^ 2)
50  HPLOT X,B + P: HPLOT X,B - P
60  NEXT X
```

17a.

```
5  REM  plot the base and pegs in green
10  HOME : GR : COLOR= 12
20  FOR X = 33 TO 35: HLIN 2,36 AT X: NEXT X
30  VLIN 20,33 AT 8
40  VLIN 20,33 AT 19: VLIN 20,33 AT 30
100  REM
110  REM  plot the 5 disks
120  REM
130  COLOR= 1
140  FOR X = 1 TO 5
150  HLIN 8 - X,8 + X AT 2 * X + 22
160  NEXT X
```

17b.

```
5  REM  plot the base and pegs in green
10  HOME : GR : COLOR= 12
20  FOR X = 33 TO 35: HLIN 2,36 AT X: NEXT X
30  VLIN 20,33 AT 8
40  VLIN 20,33 AT 19: VLIN 20,33 AT 30
100  REM
110  REM  plot the 5 disks
120  REM
130  COLOR= 1
140  FOR X = 1 TO 5
150  HLIN 8 - X,8 + X AT 2 * X + 22
155 A(1,X) = X:A(2,X) = 0:A(3,X) = 0
160  NEXT X
170  REM
180  REM  movement of disks
190  REM
200  INPUT "FROM TOWER, TO TOWER? ";T1,T2
210 DISK = 0: FOR X = 1 TO 5
220  IF DISK > 0 OR A(T1,X) = 0 THEN 260
230 DISK = A(T1,X):XCTR = 8 + (T1 - 1) * 11:A(T1,X) = 0
240  COLOR= 0: HLIN XCTR - DISK,XCTR + DISK AT 2 * X + 22
250  COLOR= 12: PLOT XCTR,2 * X + 22: COLOR= 1
260  NEXT X
270  IF DISK = 0 THEN  PRINT "NO DISK THERE!": GOTO 200
280 EMPTY = 0: FOR X = 5 TO 1 STEP  - 1
290  IF EMPTY = 0 AND A(T2,X) = 0 THEN EMPTY = X
300  NEXT X: IF EMPTY = 5 THEN 330
310  IF A(T2,EMPTY + 1) >  = DISK THEN 330
320  PRINT "YOU CAN'T DO THAT!":T2 = T1: GOTO 280
330  REM  plot the disk at A(T2,EMPTY)
340 X = 8 + (T2 - 1) * 11:Y = 2 * EMPTY + 22
350  COLOR= 1: HLIN X - DISK,X + DISK AT Y
360 A(T2,EMPTY) = DISK
370  PRINT : IF T2 <  > 1 AND EMPTY = 1 THEN 390
380  GOTO 200
390  PRINT "GOOD SHOW OL' CHAP!"
```

CHAPTER SEVEN

1.

```
10  INPUT N
20  IF N < 0 THEN  PRINT "No negative numbers allowed.": GOTO 10
30  PRINT "N=";N,"Square roots = + or -"; SQR (N): GOTO 10
```

3.

```
5 PI = 3.14159265
10  PRINT "Cos(X)"; TAB( 20);"Sin(X+90)"
20  FOR X = 0 TO 360 STEP 10
30 R = X * PI / 180: REM  Convert to radians
40  PRINT  COS (R); TAB( 20); SIN (R + PI / 2)
50  NEXT X
```

5.

```
10  INPUT N
20  PRINT  SGN (N) * N: REM  This is equivalent to ABS(N)
```

7.

```
5 PI = 3.14159265
10  PRINT "Radians","Degrees"
20  FOR A = 0 TO 3 STEP .25
30  PRINT A,A * 180 / PI
40  NEXT A
```

9.

```
5 PI = 3.14159265
10  INPUT N
15 N = N * PI / 180
20 C =  COS (N)
30 S =  SIN (N)
40 T =  TAN (N)
50  IF C > S AND C > T THEN  PRINT "Cosine =";C
60  IF T < S AND S > C THEN  PRINT "Sine =";S
70  IF T > S AND T > C THEN  PRINT "Tangent =";T
```

11.

```
10  DEF  FN F(X) = 9 * X ^ 3 - 7 * X ^ 2 + 4 * X - 1
20  INPUT "A,B? ";A,B
30  PRINT  FN F(B) -  FN F(A)
```

13.

```
10  DEF  FN R(X) =  INT (100 * X + .5) / 100
20  PRINT " X","LN(X)","EXP(X)"
30  FOR I = 1 TO 15
40  PRINT I, FN R( LOG (I)), FN R( EXP (I))
50  NEXT I
```

15.

```
]RUN
5                -4               3                4

]
```

17.

```
]RUN
FNF(-4) is negative
FNF(-2) is negative
FNF(0) is negative
FNF(2) is zero
FNF(4) is positive
FNF(6) is positive

]
```

19.

```
]RUN
-3
-2
-1
0
1
2
3
4

]
```

21.

```
5 PI = 3.14159265
10  INPUT "R and Theta (in degrees) : ";R,T
20 T = T * PI / 180
30 X = R *  COS (T)
40 Y = R *  SIN (T)
50  PRINT "(X,Y) = (";X;",";Y;")"
```

23.

```
10  FOR X = 1 TO 100
20 Y =  ABS ( SIN (X) * 1000)
30 R =  INT (Y) / 16 -  INT ( INT (Y) / 16)
40  PRINT R
50  NEXT X
```

25.

```
10  FOR N = 0 TO 6.2 STEP .2
20 U =  SIN (N)
30  PRINT  TAB( 16 * U + 17);"SHAZAM!"
40  NEXT N
```

27.

```
5  INPUT "N? ";N
10  HGR2 : HCOLOR= 5
20 X0 = 135:Y0 = 100
30  REM  Draw axes
40  HPLOT 0,Y0 TO 279,Y0
50  HPLOT X0,0 TO X0,191
60  REM  Plot Y = SIN(N*PI*X)
65  HCOLOR= 3
70  FOR X =  - 6.4 TO 6.4 STEP .2
75 X1 = X0 + X * 21
80 Y =  SIN (N * X * 3.14159)
90 Y1 = Y0 + Y * 21
100  IF (Y1 >  = 0) AND (Y1 <  = 191) THEN  HPLOT X1,Y1
110  NEXT X
```

CHAPTER EIGHT

1.

```
10  INPUT "A$? ";A$
20  FOR I = 1 TO  LEN (A$)
30  PRINT  ASC ( MID$ (A$,I,1));" ";
40  NEXT I
```

3.

```
10  READ X
20  PRINT  CHR$ (X);
30  GOTO 10
40  DATA  65,83,67,73,73,32,68,73
50  DATA  68,32,84,72,73,83,33
```

5.

```
1  REM   A$ = Original string
2  REM   L$,M$,R$ = Left,Mid,Right strings of original
3  REM
4  REM
10 A$ = "THREE!@#$%STRING!@#$%FUNCTIONS"
20 L$ =  LEFT$ (A$,5)
30 M$ =  MID$ (A$,11,6)
40 R$ =  RIGHT$ (A$,9)
50  PRINT L$;" ";M$;" ";R$
```

7.

```
1  REM   N$ = Name being examined
2  REM   A() = Number of occurrences of each letter
3  REM   S = ASCII code for the letters in A$
4  REM  F = 1 if there is a tie between letters, else F=0
5  REM
6  REM
10  REM  Get name and count occurrences
20  DIM A(26)
30  INPUT "Name? ";N$
40  FOR I = 1 TO  LEN (N$)
50 S =  ASC ( MID$ (N$,I,1))
60 A(S - 64) = A(S - 64) + 1
70  NEXT I
80  REM
90  REM
100  REM   Find most common letter
110  REM
120 M = 1
130  FOR I = 1 TO 26
140  IF A(I) = M THEN F = 1
150  IF A(I) > M THEN M = A(I):V = I:F = 0
160  NEXT I
170  REM
180  REM
200  REM   Print the most common letter
210  IF F = 0 THEN  PRINT  CHR$ (V + 64);" is the most com
     mon letter.": END
220  PRINT "The following letters have an"
230  PRINT "equal occurrence:"
240  FOR I = 1 TO 26
250  IF A(I) = M THEN  PRINT  CHR$ (I + 64);" ";
260  NEXT I
```

9.

```
10  READ W$
20  FOR I = 1 TO  LEN (W$)
30 T$ =  MID$ (W$,I,1)
40  IF T$ <  > "E" THEN  PRINT T$;
50  NEXT I
60  PRINT : GOTO 10
100  DATA  QUEEN,LENGTH,REMEMBER
```

11.

Binary	Decimal
1011	11
10100	20
1111	15
1110	14
1010011	83
110011	51
1011100	92
1101111	111
11000000	192
10000111	135

13.

```
10 A$ = "ABCDEFGHIJKL"
20  FOR I = 1 TO 12
30  PRINT  LEFT$ (A$,I)
40  NEXT I
```

15.

```
]RUN
87
72
55
54

]
```

17.

```
10  FOR X = 1 TO 15
20 I =  INT (26 *  RND (1) + 65)
30  PRINT  CHR$ (I);
40  NEXT X
```

19.

```
10  FOR X = 1 TO 100
20 A$ = A$ +  CHR$ ( INT (95 *  RND (1) + 32))
30  NEXT X
40  PRINT : PRINT A$: PRINT
50  FOR Y = 1 TO 100
60 C =  ASC ( MID$ (A$,Y,1))
70  IF (C > 64 AND C < 91) THEN L = L + 1: GOTO 100
80  IF (C > 47 AND C < 58) THEN N = N + 1: GOTO 100
90 M = M + 1
100  NEXT Y
110  PRINT "There are ";L;" letters, ";N
120  PRINT "Numbers and ";M;" misc. characters."
```

21a.

```
10  INPUT "Message to encode? ";M$
20  FOR I = 1 TO  LEN (M$)
30 C =  ASC ( MID$ (M$,I,1))
40  IF C = 32 THEN 80
50  IF C < 65 OR C > 90 THEN 100
60 C = C + 2
70  IF C > 90 THEN C = C - 26
80 A$ = A$ +  CHR$ (C)
90  NEXT I
100  PRINT "Encoded message: "
110  PRINT A$
```

21b.

```
10  INPUT "Message to decode? ";M$
20  FOR I = 1 TO  LEN (M$)
30 C =  ASC ( MID$ (M$,I,1))
40  IF C = 32 THEN 80
50  IF C < 65 OR C > 90 THEN 90
60 C = C - 2
70  IF C < 65 THEN C = C + 26
80 A$ = A$ +  CHR$ (C)
90  NEXT I
100  PRINT "Message:"
110  PRINT A$
```

23.

```
1  REM   T$ = A word from the sentence
2  REM   W$() = Up to 30 word sentence
3  REM   S$ = Original sentence
4  REM
5  REM
10  DIM W$(30)
20  INPUT "Your sentence? ";S$
30 S$ =  CHR$ (32) + S$ +  CHR$ (32)
40  FOR X = 1 TO  LEN (S$)
50 T$ =  MID$ (S$,X,1)
60  IF  ASC (T$) = 32 THEN C = C + 1: GOTO 80
70 W$(C) = W$(C) + T$
80  NEXT X
90  FOR L = 30 TO 1 STEP  - 1
100  IF W$(L) <  > "" THEN  PRINT W$(L);" ";
110  NEXT L
```

25.

```
10  INPUT "Enter a binary number ";N$
20  FOR N = 1 TO  LEN (N$)
30 D = D +  VAL ( MID$ (N$,N,1)) * 2 ^ ( LEN (N$) - N)
40  NEXT N
50  PRINT D
```

27.

```
10  DIM S$(30),N(9)
20 C = C + 1
30  PRINT "Sentence? ";
40  GET A$: PRINT A$;
45  IF  ASC (A$) = 13 THEN 70
50 S$(C) = S$(C) + A$
60  GOTO 40
65  REM  Receive sentences until a sentence is of zero length
70  IF  LEN (S$(C)) = 0 THEN 100
75  REM  Receive sentences until a sentence is of zero length
80 S$(C) =  CHR$ (32) + S$(C) +  CHR$ (32)
90  GOTO 20
100  FOR L = 1 TO C - 1
110  RESTORE
120  FOR W = 1 TO 9
130  READ T$
140  GOSUB 300
150 N(W) = N(W) + R
160  NEXT W
170  NEXT L
180  PRINT "Articles: ";N(1) + N(2) + N(3)
```

29.

```
190  PRINT "Adverbs: ";N(4)
200  PRINT "Punctuation marks: ";N(5) + N(6) + N(7) + N(8) + N(9)
210  END
220  REM
230  REM
300 R = 0
310  FOR P = 1 TO  LEN (S$(L))
320  IF  MID$ (S$(L),P, LEN (T$)) = T$ THEN R = R + 1
330  NEXT P
340  RETURN
400  DATA  " A "," AN "," THE ","LY ",".","!","?",";",","
```

CHAPTER NINE

1a.

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  DIM R(50)
30  FOR X = 1 TO 50
40 R(X) =  INT (20 *  RND (1) + 1)
50  NEXT X
60  PRINT DOS$;"OPEN RANDOM NUMBERS"
70  PRINT DOS$;"WRITE RANDOM NUMBERS"
80  FOR I = 1 TO 50
90  PRINT R(I)
100  NEXT I
110  PRINT DOS$;"CLOSE RANDOM NUMBERS"
120  END
```

1b.

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN RANDOM NUMBERS"
30  PRINT DOS$;"READ RANDOM NUMBERS"
40  FOR I = 1 TO 50
50  INPUT R
60 S = S + R
70  NEXT I
80  PRINT DOS$;"CLOSE RANDOM NUMBERS"
90  PRINT "The sum of the numbers is ";S
100  END
```

3a.

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  FOR X = 1 TO 10
30  INPUT "Name? ";N$(X)
40  NEXT X
50  PRINT DOS$;"OPEN FIRST NAMES"
60  PRINT DOS$;"WRITE FIRST NAMES"
70  FOR I = 1 TO 10
80  PRINT N$(I)
90  NEXT I
100  PRINT DOS$;"CLOSE FIRST NAMES"
110  END
```

3b.

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20 N = 0: REM  Set name counter
30  PRINT DOS$;"OPEN FIRST NAMES"
40  PRINT DOS$;"READ FIRST NAMES"
50  FOR P = 1 TO 10
60  INPUT N$
70  IF N$ > "C" AND N$ < "I" THEN  PRINT N$:N = N + 1
80  NEXT P
90  PRINT DOS$;"CLOSE FIRST NAMES"
100  IF N = 0 THEN  PRINT "None found"
110  END
```

5a.

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  FOR X = 1 TO 10
30  INPUT "Title? ";T$(X)
40  NEXT X
50  PRINT DOS$;"OPEN BOOKS,L60"
60  FOR I = 1 TO 10
70  PRINT DOS$;"WRITE BOOKS,R";I
80  PRINT T$(I)
90  NEXT I
100  PRINT DOS$;"CLOSE BOOKS"
110  END
```

5b.

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  PRINT DOS$;"OPEN BOOKS,L60"
30  PRINT DOS$;"OPEN N-Z TITLES"
40  FOR B = 1 TO 10
50  PRINT DOS$;"READ BOOKS,R";B
60  INPUT T$
70  IF T$ < "N" THEN 100
80  PRINT DOS$;"WRITE N-Z TITLES"
90  PRINT T$
100  NEXT B
110  PRINT DOS$;"CLOSE BOOKS"
120  PRINT DOS$;"CLOSE N-Z TITLES"
130  END
```

7a.

```
100  REM  Data input section
110  REM
120 DOS$ =  CHR$ (4): REM  CTRL-D
130  DIM F(12),M(12),C(12)
140  FOR A = 1 TO 12
150  PRINT "MONTH #";A
160  INPUT "FULL-SIZED? ";F(A)
170  INPUT "MID-SIZED? ";M(A)
180  INPUT "COMPACT? ";C(A)
190  PRINT
200  NEXT A
210  REM
220  REM
300  REM  Data output section
310  REM
320  PRINT DOS$;"OPEN CARS,L20"
```

```
330  FOR B = 1 TO 12
340  PRINT DOS$;"WRITE CARS,R";B
350  PRINT F(B): PRINT M(B): PRINT C(B)
360  NEXT B
370  PRINT DOS$;"CLOSE CARS"
380  END
```

7b.

```
100 DOS$ =  CHR$ (4): REM  CTRL-D
110  HOME
120  PRINT DOS$;"OPEN CARS,L20"
130  PRINT "Enter a negative number to stop"
140  INPUT "Which month? ";M1
150  IF M1 < 0 THEN 370
160  PRINT DOS$;"READ CARS,R";M1
170  INPUT F: INPUT M: INPUT C
180  PRINT DOS$
190  REM
200  REM  Bar graph section
210  REM  Each * represents one car
220  REM
230  PRINT "F: ";
240 L = F: GOSUB 300
250  PRINT "M: ";
260 L = M: GOSUB 300
270  PRINT "C: ";
280 L = C: GOSUB 300
290  GOTO 140
300  FOR A = 1 TO L
310  PRINT "*";
320  NEXT A
330  PRINT : PRINT
340  RETURN
350  REM
360  REM
370  PRINT DOS$;"CLOSE CARS"
380  END
```

9a.

```
100 DOS$ =  CHR$ (4): REM  CTRL-D
110  REM  Build Sheryl's file
120  REM
130  PRINT "Enter Sheryl's data"
140  FOR A = 1 TO 6
150  INPUT "Time? ";H(A)
160  INPUT "Temperature? ";F(A)
170  PRINT
180  NEXT A: PRINT
190  PRINT DOS$;"OPEN SHERYL"
200  PRINT DOS$;"WRITE SHERYL"
210  FOR B = 1 TO 6
220  PRINT H(B): PRINT F(B)
230  NEXT B
240  PRINT DOS$;"CLOSE SHERYL"
250  REM
260  REM
300  REM  Build Carol's file
```

```
310  REM
320  PRINT "Enter Carol's data"
330  FOR C = 1 TO 6
340  INPUT "Time? ";H(C)
350  INPUT "Temperature? ";F(C)
360  PRINT
370  NEXT C
380  PRINT DOS$;"OPEN CAROL"
390  PRINT DOS$;"WRITE CAROL"
400  FOR D = 1 TO 6
410  PRINT H(D): PRINT F(D)
420  NEXT D
430  PRINT DOS$;"CLOSE CAROL"
440  END
```

9b.

```
10 DOS$ =  CHR$ (4): REM  CTRL-D
20  DIM T(100)
30  PRINT DOS$;"OPEN SHERYL"
40  PRINT DOS$;"OPEN CAROL"
50  PRINT DOS$;"OPEN MERGE"
60  FOR L = 1 TO 6
70  PRINT DOS$;"READ SHERYL"
80  INPUT H: INPUT F
90  GOSUB 400
100  PRINT DOS$;"READ CAROL"
110  INPUT H: INPUT F
120  GOSUB 400
130  NEXT L
140  REM
150  REM
200  REM  Build MERGE
210  REM
220  PRINT DOS$;"WRITE MERGE"
230  FOR Z = 1 TO 100
240  IF T(Z) <  > 0 THEN  PRINT Z / 10: PRINT T(Z)
250  NEXT Z
260  PRINT DOS$;"CLOSE SHERYL"
270  PRINT DOS$;"CLOSE CAROL"
280  PRINT DOS$;"CLOSE MERGE"
290  END
300  REM
310  REM
400  IF T(H * 10) = 0 THEN T(H * 10) = F: RETURN
410 T(H * 10) = (T(H * 10) + F) / 2: RETURN
420  REM
430  REM  The times for each measurement are multiplied
440  REM  by ten and the resulting integers are used
450  REM  as subscripts in a large one dimensional array.
460  REM  Simultaneous entries are averaged.
```

9c.

```
10 DOS$ =  CHR$ (4): REM  CRTL-D
20  ONERR  GOTO 80
30  PRINT DOS$;"OPEN MERGE"
40  PRINT DOS$;"READ MERGE"
50  INPUT H: INPUT F
60  PRINT H,F
70  GOTO 50
80  PRINT DOS$;"CLOSE MERGE"
90  END
```

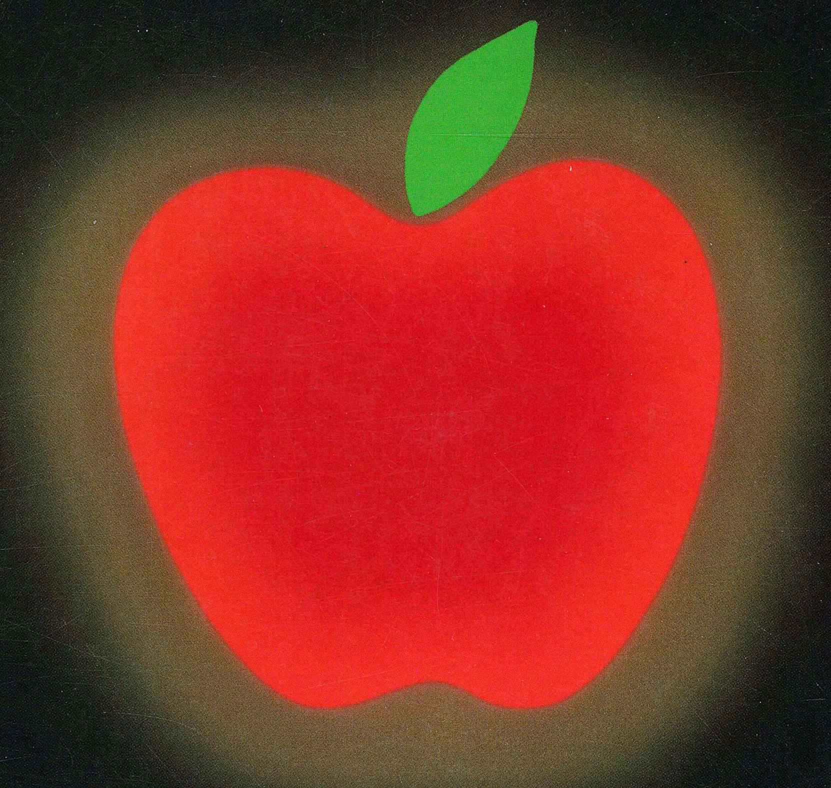
Index

ABS 7.1
Algorithms 5.6
Animation 6.15
AND 2.5
APPEND 9.6
Arccosine 7.4
Arcsine 7.4
Arctan 7.4
ASC 8.5
ASCII Character Conversion 8.5
ASCII Code 8.5
ASCII table 8.3
ATN 7.4
Base ten 8.1
Base two 8.1
BASIC ix
Binary code 8.1
Bit 8.2
Byte 8.2
Cassette Files B.1
Cassette Operation B.1
CATALOG C.2
CHR$ 8.5
CLEAR 4.9
CLOSE 9.3
COLOR 6.2
Comparing numeric variables 2.2
Comparing strings 2.3
Computer Languages x
Computer Memory and Processing 8.1
Conditional 2.1
Conditional, extended use 2.3
Conversion, degrees, radians 7.3
CONT 5.19
Correcting errors 1.2, 5.12
COS 7.3
CONTROL key 1.10
CONTROL-C 1.10
CONTROL-D 9.1
CONTROL-S A.1
CONTROL-X A.1
Cursor movement A.1
DATA 1.7
Data Types 8.4
Debugging 5.12
DEF 7.6
DELETE C.3
Degree Conversion 7.3
Development of the computer, The ix
DIM 4.9
Diskette directories C.2
Diskette Files C.1
Diskette formatting C.1
Double Subscripted Variables 4.10
e^x 7.5
Editing A.1
END 2.6
Error, ?OUT OF MEMORY 4.9
Error, ?REMIM'D ARRAY 4.9
Errors, correcting 1.2, 5.12
Errors, Logic 5.13
Errors, Rounding 3.4
Errors, Runtime 5.12
Errors, Typing 1.2
Errors, syntax 5.12
ESC key A.1
EXP 7.5
Exponential Function 7.5
Exponentiation 2.1, 2.9
Extended Use of the
  IF . . . THEN statement 2.3
Extended Variable Names 4.13
File names C.2
Files, cassette B.1

Files, diskette 9.1
FLASH 3.6
FOR . . . TO . . . STEP 2.7
Format of File Commands 9.1
Formatting Diskettes C.1
Game Programs 3.1, 3.9
GET 3.8
GOSUB 5.1, 5.4
GOTO 1.7, 5.4
GR 6.1
Graphics 6.1
Hand Tracing 5.17
HCOLOR 6.10
HGR 6.10
HGR2 6.10
High resolution graphics 6.9
HLIN 6.3
HOME 3.6, 6.1
HPLOT 6.10
HTAB 3.6
IF . . . THEN 2.1
Immediate Mode 1.14
Indenting 5.8
INIT C.1
INPUT 1.9
INT 3.1
INVERSE 3.6
LEFT$ 8.6
LEN 8.6
Line Numbering 1.1
LIST 1.2
LOAD C.2
LOCK C.3
Logarithms 7.5
LOG 7.5
Logic Errors 5.13
Loops 2.7, 4.1
Low resolution graphics 6.1
MAXFILES C.4
MID$ 8.6
Multiple Drive Systems C.4
Multiple File Use 9.22
Multiple Statement Lines 2.3
Need for Subscripted Variables 4.3
Nested FOR . . . NEXT Loops 4.1
NEW 1.3
NEXT 2.7
NORMAL 3.6
NOTRACE 5.18
Numeric Variables 1.3
ON . . . GOSUB 5.4
ON . . . GOTO 5.2
ONERR GOTO 5.14
OPEN 9.2, 9.14
OR 2.5
Order of Operations 2.1
PEEK 10.6
Pi (π) 7.3
Planning a Program 5.5
PLOT 6.2
POS 3.5
PRINT 1.1
PRINT Formatting 1.10
PRINT Punctuation 1.11
PRINT TAB 1.12
Printing zones 1.11
Program and file names C.2
Program structuring rules 5.8
Proper Care of Equipment (diskettes) C.1
Quotation marks 1.2
Radian conversion 7.3
Raising to the power 2.1, 2.9
Random-Access Files 9.13
Random numbers 3.1
READ 1.7, 9.5, 9.16
RECALL B.1
REM 1.15
Removing Programs and Files,
  diskette C.3
RENAME C.2
Renaming Programs and Files C.2
RESTORE 2.10
RESUME 5.14
RETURN 5.1
RETURN key 1.1
RIGHT$ 8.6
RND 3.1
Rounding Error 3.4
Runtime errors 5.12
SAVE C.1
Scientific notation 1.7
Screen dimensions 1.11, 1.12, 6.6, 6.9
SCRN 6.6
SGN 7.1
Shifting and Scaling a Graph 7.8
SIN 7.3

Single Subscripted Variables 4.4
Some Final Notes on
  Subscripted Variables 4.13
SPEED 4.1
Switching Displays 6.13
SQR 7.1
Starting Programs at
  Arbitrary Locations A.2
STEP 2.7
STOP 5.19
Storing Strings on Tape B.2
STORE B.1
STR$ 8.6
String Manipulation 8.6
String Variables 1.5
Structuring a program 5.7
Subroutines 5.1
Subscripts 4.4
Subscripted variables 4.3
Summation 3.3
TAB 1.12
TAN 7.3
TEXT 6.1
TRACE 5.18
Tracing 5.14
Trigonometric Functions 7.3
Truncating decimals 3.1
Typing Errors 1.2
Unacceptable variable names 4.13
UNLOCK C.3
Updating Sequential Files 9.6
Updating Random-Access Files 9.19
User defined functions 7.6
VAL 8.6
Variables 1.3
Variables, extended names 4.13
Variables, names 1.3
Variables, Numeric 1.3
Variables, String 1.5
Variables, subscripted 4.3
VLIN 6.3
VTAB 3.6
What is a computer? ix
What is a program? ix
What is a variable? 1.3
WRITE 9.3, 9.14
Zones, printing 1.11

FPT >$16.50



A GUIDE TO PROGRAMMING IN APPLESOFT is a comprehensive manual designed to help student programmers exploit thoroughly the considerable capabilities of their Apple IIe and II+ computers. Appropriate for people with a wide range of backgrounds, this manual is structured in a unique way. Topics are discussed individually so that students may select the sequence that interests them. Individual chapters are devoted to programming techniques, color graphics, mathematical functions, data types, and the disk operating system.

Although the manual has been developed and extensively tested for classroom use, it is also an effective tool for individual study. From the beginning of the manual, students are taught good programming techniques which are essential to writing logical, easy to read programs. This manual also features review exercises in each chapter and comprehensive problem sets, which offer the student programmer an opportunity to try out newly learned skills. Solutions to odd-numbered problems are given at the back of the manual.

Bruce Presley, a graduate of Yale University, has taught computer programming at The Lawrenceville School for more than fifteen years and has served on the founding committee of the Advanced Placement Computer Science Committee of the College Entrance Examination Board.

Other titles in this best-selling series include:

A GUIDE TO PROGRAMMING THE COMMODORE COMPUTERS
A GUIDE TO PROGRAMMING THE IBM PERSONAL COMPUTER
A GUIDE TO PROGRAMMING IN LEVEL II BASIC (Radio Shack)
A GUIDE TO PROGRAMMING IN BASIC-PLUS (Digital)

ISBN 0-442-27249-9

VAN NOSTRAND REINHOLD COMPANY
NEW YORK CINCINNATI TORONTO LONDON MELBOURNE

BRUCE PRESLEY

A GUIDE TO PROGRAMMING IN APPLESOFT

SECOND
EDITION

VAN NOSTRAND
REINHOLD

VNR